fetlines™

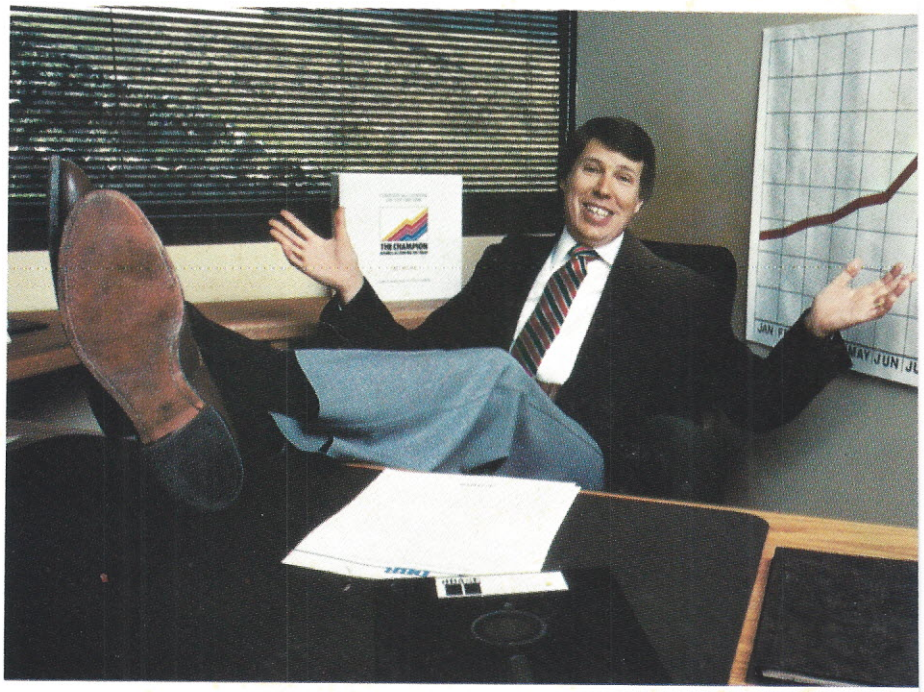
The Software Magazine™

Volume IV, No. 8 (ISSN 02479-2575, USPS 597-830)

$3.00 January 1984

"dBASE II® is far, far better than a shoehorn."

Rusty Fraser
President
Data Base Research Corp.



"We laughed when our customers asked us to put our minicomputer-based real-time accounting system, The Champion,™ on a micro.

"No way was it going to fit, we thought.

"We'd have to create our own database management system and, even then, it'd be a tight squeeze.

"Then we discovered dBASE II, the relational database management system for microcomputers from Ashton-Tate."

"dBASE II was a perfect fit."

"dBASE II is a program developer's dream come true. The dBASE II RunTime™ module quickly provided us with the powerful text editing, data entry speed and other 'building block' capabilities we needed to develop and deliver a new Champion to our customers—the leading real-time on-line accounting system available for a micro."

The short cut to success.

The dBASE II RunTime module opers like Data Base Research become successful software publishers.

For more about dBASE II and RunTime, contact Ashton-Tate 10150 West Jefferson Boulevard, Culver City, CA 90230, (800) 437-4329, ext. 217. In the U.K., call (0908) 568866.

For more about The Champion, call Data Base Research at (303) 987-2588.

ASHTON·TATE

Lifelines
The Software Magazine

Publisher: Edward H. Currie
Editor in Chief: Susan E. Sawyer
Production Manager: Kate Gartner
Technical Editor: Al Bloch
Art Director: Kate Gartner
Typographer: Rosalee Feibish
Cover: Kate Gartner

Dealer/Customer Service Manager: William F. Lampe
Circulation Manager: Trina McDonald
Customer Service Assistant: Robert Laing
Advertising Manager: William F. Lampe
New Versions Editor: Lee Ramos
Production Assistant: Marcy Rauch
Printing Consultant: Sid Robkoff/E&S Graphics

Lifelines (ISSN 0279-2575,USPS 597-830) is published monthly at a subscription price of $24 for twelve issues, when destined for the U.S. Canada, or Mexico, $50 when destined for any other country. Second-class postage at New York, New York. POSTMASTER, please send changes of address to Lifelines Publishing Corporation, 1651 Third Avenue, New York, NY 10028.

Editorial

Omar would have been proud . . .

by Edward H. Currie

If ever there was an industry for which the poetry of Omar Khayyam was ideally suited, it is the microcomputer industry:

"The moving finger writes; and, having writ, moves on. . . ."

" 'Tis all a checkerboard of nights and days where destiny with men for pieces plays: hither and thither moves, and mates, and slays, and one by one back in the closet lays. . . ."

Rubaiyat
Omar Khayyam

The COMDEX show in Las Vegas was interesting, but I now know what the limiting factor in the micro world is. . . Human Endurance. A hundred years from now when our great grandchildren ask why the computer industry never got any bigger the answer will undoubtedly be that no one had the physical strength or the emotional stamina to visit even one more booth.

Estimates varied from 11 to 17 miles of aisles and some 11 to 1400 exhibitors. Everything from 9-track tape drives to print wheels was seen and in great abundance and redundancy.

Hardware manufacturers had, for the most part, little to offer us that was either new or, if new, likely to be of interest at the next COMDEX show.

LCD micros abounded; but does anyone seriously believe that 4 to 16 lines of 64 characters will be of any interest other than historical when the technology advances to 24x80 and beyond? Save your confederate and yankee dollars. . . there are better and more lasting buys on the way.

One exceptional entry is Radio Shack's Model 2000. Gone is the yucky gray plastic of the Trash-80 and in its place is a stylish white cabinet housing a variety of very interesting goodies. This 80/86 based machine offers 1.4 Mbytes of disk storage, 128K RAM (standard), maximum RAM of 768K, 8 Mhz clock speed, four expansion slots (not IBM compatible), MS-DOS 2.0, RS-232/Parallel ports, and graphics options galore. The graphics supported is 640x400 for both monochrome and color (eight colors) and graphics on monochrome display is also supported. Optional 10 Mbyte hard disk, mice and joysticks are also offered.

Microsoft GW-BASIC, FORTRAN and PASCAL compilers are offered as well as Microsoft's assembler. It has been announced that Microsoft Windows will also be available and several of the more popular application packages such as dBASE II are compatible. Interestingly, MAI/Basic Four's accounting packages provide general ledger, accounts payable, accounts receivable, inventory control, order entry and purchase orders.

The basic cabinet styling of this machine is similar to that of the IBM-PC but the really interesting aspects of this machine are the object code compatibility with the 8088/8086, 8 Mhz clock, 640x400 color graphics, 640x400 monochrome graphics, 16-bit data path, and MS-DOS 2.0 support. The color graphics monitor provides some of the best quality graphics I have seen for micros.

This will do much to inject life into MS-DOS on non-IBM machines. Note that while this system will read IBM diskettes not all IBM software will be compatible, e.g., screen graphics is handled differently in the two machines at the hardware level. It appears that Radio Shack has made a quantum leap forward in their product offerings with the introduction of the Model 2000. And there's a real opportunity for applications writers to take advantage of this machine's disk capacity, speed and impressive graphics capability. A number of window products were offered at COMDEX including Lattice Window for C applications writers, Microsoft "Windows," Digital Research's latest — Concurrent CP/M with Windows, Visi Corps Visi On, Softech's Windows for UCSD P System, Des Q, etc.

And, of course, the usual cadre of self-styled, self-appointed visionaries, analysts, gurus and pontificants were on hand at COMDEX. Particularly disappointing was one panel which advised their audience that unless you the author were as well financed as Lotus 123 your product would be "lost in the noise." This type of nonsensical declaration does nothing to stimulate the development of new and exciting software. It is sad that in their rush to serve their own needs for recognition they are given to the propagation of such sheer rubbish. Few, if any, of the more successful programs available today were provided by major corporations. Typically, only when software authors took the initiative and developed programs in their "basements" has much technological progress been made. What of people such as Francis Lynch — author of Lattice C, or Peter Rosien — author of T-Maker, or Bill Gates/Paul Allen — who gave us BASIC, the most popular microcomputer language in the world, or Bricklin's Visa Calc, or Mitch Kapor/Jonathan Sach's 123, or CP/M, written by Gary Kildahl while a consultant for Intel (who, incidentally, turned down CP/M), or Gordon Eubanks — author of CBASIC which was written on the floor of a submarine, or Tim Paterson, who single-handedly wrote MS-DOS, or any of the litany of the most important software products available today.

It is certainly true that the cost of marketing products on a worldwide basis has risen substantially in recent months but there are plently of software publishers, book publishers, distributors and marketing organizations anxious to take your product and focus the market's attention on it.

My concern is not for the misguided few who are telling you to forget it, they may mean well but they don't really have the scope to warrant much long term attention. But rather that the independent authors around the world who are a source for the ever increasing demand for high quality software not become disenchanted or disenfranchised. Write the best applications that you can and the world will beat a path to your door. As for the pontificants, if you are uncertain as to how much credence to give such people go the failsafe route and ignore them. . . the only way that they can harm you is if you listen. . . .

Rana and PCPI have both introduced 8086 and 8088 cards respectively for the Apple and both support MSDOS. Lattice C is currently available on the PCPI board and the system is incredibly fast compared to what is typically to be expected from the Apple.

In an industry based on buzz words and worn out phrases I suppose one shouldn't be shy about coining new terms. David Bunnell is the father of the term "Vaporware" so I decided that my contribution would be "Ether-ware." Both of these terms refer to software/hardware with the first referencing software/hardware that never get beyond the announcement stage. The second term refers to software/hardware that never exists in any form more substantive than the ether. Note that both are treated as real. There is allegedly a continuum beginning with etherware progressing to vaporware and then finally to software/hardware.

All of these "-wares" are the province of the trade press and you should "be-ware" of being mislead by claims for all of the latest and greatest software packages and hardware devices, particularly those which offer complex features described as

(continued on page 35)

Review

SuperCalc 2

by Joseph B. Rothstein

Electronic spreadsheet programs have probably been a motivating factor behind the purchase of more computers than any other single piece of software. But for those who may still be unaware of this ubiquitous application, a spreadsheet is a two-dimensional grid containing cells at the intersection of each row and column. The user may enter information into any of the cells or, using commands associated with the spreadsheet program, describe relationships between cells or groups of cells. The value within a cell may result from computations based on the contents of other cells. Changes to the values of just a few cells can influence the entire spreadsheet, expediting the process of modeling or "what-if" analysis of different conditions or options.

The first spreadsheet for microcomputers was VisiCalc, for the Apple II, and it spawned a host of imitators intended for a variety of machines. One of the earliest "VisiClones" was SuperCalc from Sorcim Corporation, designed for microcomputers running the CP/M operating system.

Like VisiCalc, SuperCalc implemented the electronic spreadsheet quite literally, with many of the limitations inherent in the paper-and-pencil version. These limitations included restrictions on cell size and flexibility, text-manipulation operations, calculation speed, and integration of information from different spreadsheets into a single data structure.

Sorcim has addressed these limitations with a new version, called SuperCalc-2, and it matches its popular predecessor in documentation and ease of use, while surpassing it in performance, versatility and speed. It is one of several "second-generation" spreadsheet programs which more fully utilize the advantages a computer has over a paper-based spreadsheet. It also appears to be among the best.

SuperCalc - 2 is available for a variety of microcomputers running the following operating systems: CP/M, MP/M, CP/M-86, MP/M-86, PC-DOS, and MS-DOS. For this review I used a homebrew S-100 system running version 2.2 of CP/M-80.

Installation

A 22-page document covers all aspects of installation, beginning with how to make a copy of the distribution diskette, and continuing on to the heart of the installation procedure — terminal selection and printer set-up. The distribution diskette label indicates whether the program requires installation. If so, the user must run an INSTALL program supplied with the package. It prints a list of terminals supported, including most popular brands, and the user simply chooses the appropriate one. If the particular terminal used is not on the list, the printed documentation suggests compatible choices for many additional brands. If neither of these approaches handles your particular terminal, you might have to contact your dealer or Sorcim Corporation for help. But judging by the list of terminals and compatibles, this should be a rare problem.

A limited degree of printer support is also provided in the INSTALL program procedure. The user may choose settings or use the defaults governing the printer page dimensions, optional page pause, single or double spacing, CR or CR/LF pair, border character, and an optional printer initialization string. Many of these may also be set during the operation of SuperCalc-2 itself.

Features

SuperCalc-2 operates in three distinct modes. Upon entering the program, the user is automatically in Spreadsheet mode. The spreadsheet cursor is active and the edit cursor is inactive. You can move the cursor around the spreadsheet to view cell contents and values.

From Spreadsheet mode, the user can enter Data Entry mode or Command mode. You cannot go directly between Data Entry and Command mode; rather, you must return to Spreadsheet mode first. Changing modes is such a trivial task, though, that it will quickly become second nature.

Data Entry mode allows the user to enter data into a particular cell. This data may consist of text, indicated by beginning with a double quote ("), repeating text, indicated by a leading single quote ('), or a formula, which may consist of a numeric constant, a reference to another cell, a math, calendar, or special function, or a text constant. An in-line editor allows rudimentary control over the input line and simplifies data entry and correction.

It is one of several "second-generation" spreadsheet programs which more fully utilize the advantages a computer has over a paper-based spreadsheet.

Command mode entries direct the program to perform a specified action, and on the basis of variety, easy entry and usefulness of these commands, SuperCalc-2 excels. Commands begin with the slash (/) character, followed by a letter and one or more options. Entering just the slash causes the prompt line to display all the possible one letter entries. Once the user enters the first letter, the interpretive prompt completes the command. Thus, if a user enters "/B",

the command line will automatically display "/Blank". The prompt line also displays the options available at any command level. Further information about the available options may be obtained at any time by pressing the "?" key.

The /W (Window) command splits the display into two separate windows, horizontally or vertically, rather than a single, larger display. Each window may be scrolled and manipulated individually or in a synchronized fashion.

Other commands facilitate consolidation of multiple spreadsheets, automatic execution of instructions stored in a disk file, arranging row or column entries in ascending or descending order, and numerous formatting options which can control the appearance of the spreadsheet from the level of a single cell to the global level.

Recalculation of the spreadsheet may be performed automatically or manually forced, and the order of recalculation may be specified. All calculations begin with cell A1, but the user may specify that each row be calculated left to right before moving to the next row, or each column may be calculated top to bottom before moving to the next one.

Recalculation, formatting, and sorting operations perform faster than they did in the original SuperCalc. The delay they cause is still noticeable, but easily within the limits of my tolerance. Execution speed should only become a consideration while processing unusually large or complex models.

SuperCalc-2 incorporates numerous functions and operators, including some not found on its predecessor. Available functions include a wide range of arithmetic operators, transcendentals, roundoffs, averages, logical operators, conditional branches, date manipulations, and Net Present Value, among others.

SuperData Interchange (SDI) is a separate utility program supplied with the SuperCalc-2 package. It converts data stored in one disk storage format into another file with a different format. Thus, SDI provides a way to transfer SuperCalc-2 data to and from comma-delimited data files used by such programming languages as BASIC or Pascal, other applications programs including dBASE II and DataStar, and programs supporting the Data Interchange Format (DIF) defined by Software Arts, Inc. and used in the Visi- series of programs. It appears to work as specified, and may prove to be of considerable value to those who interchange data files, passing values among programs or sharing data files with other computers.

Documentation

The SuperCalc-2 distribution diskette is accompanied by several documents, in several physical formats and with a distinct audience in mind. The documentation includes Version 1.0 of the Users' Guide and Reference Manual for SuperCalc-2 and Version 1.1 of the Users' Guide and Reference Manual for SuperData Interchanger, a utility program supplied with the package. Both of these are on 8½x11-inch sheets in a three-ring notebook binder. Tucked in a pocket of the notebook are an Installation Guide (in the same format as the manuals), the license agreement and registration card, a smaller, spiral-bound booklet called "10 Minutes to SuperCalc-2," a pocket-sized Quick Reference Card, and several labels for diskettes.

Experienced computer users may choose to ignore the tutorial altogether, concentrating on the reference guide, quick reference card, and on-line help.

I began with the "10 Minutes . . ." booklet. In 20 pages of oversized type, it briefly explains what a spreadsheet is, how to invoke the program, what cells are and how the cursor is used to access them, how to enter or change the contents of a cell, how to save or reload your work, and so on. All of this material is covered in greater detail in the tutorial portion of the reference manual. A neophyte user who is baffled by computers and afraid of making mistakes might use the booklet to get his or her feet wet and get over the hurdle of intimidation. Anyone else is likely to find it over-simplified and too limited in scope.

Roughly half the material in the three-ring binder is devoted to a more complete tutorial introduction to SuperCalc-2. Like the introductory booklet, it is oriented toward the rank beginner and assumes no prior experience with computers.

Following a brief introduction and a section on "Getting Started," it proceeds to a series of twelve lessons designed to teach the basic skills needed to use SuperCalc-2. These are presented in a logical, incremental fashion using example spreadsheet models throughout, drawing on and reinforcing the skills learned in previous lessons. Unfortunately, the text sometimes begin to sound like an outtake from "Mister Rogers' Neighborhood" as in the following verbatim quote: "Let's shift the text on row 1 so that all text entries are right justified. Can you do it? Of course you can." Hand-holding is all well and good, but there are limits to everything. Yet despite such occasional condescension, the lessons do a good job of introducing the concepts and use of SuperCalc-2. I would rather become impatient with a tutorial than be confused by it.

The second half of the manual serves as a reference for the more experienced user. Such topics as the cursor, cells, operation modes, slash commands and formulas are covered in a more straightforward, succinct, and thorough manner than in the tutorial. Experienced computer users may choose to ignore the tutorial altogether, concentrating on the reference guide, quick reference card, and on-line help.

Finally, the manual contains several appendices, including a list of error messages (apparently complete), and a description of the condition that generates each error. Other appendices discuss file handling, offer a glossary of terms, and display the ever-present ASCII chart. A reasonably complete and accurate index completes the primary printed documentation.

On-line help deserves special mention, because it is implemented in an exemplary fashion. Help messages are easily summoned by the user, they are brief and to the point, and

they readily serve to refresh the memory or indicate possible actions in most cases involving routine activities. I suspect that given only the quick reference card and help messages, an experienced user could still learn the system without great difficulty.

Support

I'm always distressed when software documentation doesn't include the manufacturer's phone number along with the name and address. Alas, this is the case with SuperCalc-2. So far, I haven't needed to bend Sorcim's ear. But I'd still be happier if I had that option, because I might need it in the future. Many software manufacturers maintain a technical advice hotline, and some even have toll-free numbers for licensed users.

The Supercalc-2 manual contains no bug report forms or information regarding upgrades, newsletters or other services. More and more manufacturers are realizing that support is the strongest deterrant for piracy, and it's surprising that Sorcim isn't more sensitive to this fact. Support keeps customers happy, and the software world needs more of it.

Conclusions

SuperCalc opened up the world of spreadsheets to a great many CP/M users, and enjoyed widespread popularity. The new version is noticeably improved, and compares favorably to the most powerful of its more recent competitors. With the second generation, spreadsheets have become such a competitive application that *de facto* standard features have begun to emerge. The choice of one particular program over another may as justifiably be based on such factors as documentation, ease of use, or integration with other products as on whether it includes every conceivable bell and whistle.

Another spreadsheet program might have a slightly different combination of features than SuperCalc-2, making it more appropriate for solving a particular specialized problem. But SuperCalc-2 appears equal to any but the most unusual demands. Spreadsheets have already become part of our business and financial culture, and SuperCalc-2 represents a significant step forward in their evolution.

Spreadsheets still sell computers, and the unassuming approach and tutorial slant evident throughout SuperCalc-2 should appeal to those who may have doubts about what they're getting into. Its combination of features and ease of use should appeal to new and experienced users alike.

Review

Organizer II

by Bob Kowitt

A front end to help you relax on your back end

Would you like to enable your new client to use his new CP/M system without the need to call you every few minutes? Would you like to free yourself of the need to break away to hold your client's hand every time you develop a new application that may require any series of operations beyond the basic RUN mode? The answer may be in a new version of a CP/M front end program called ORGANIZER II, $149, from The Information People of Newark, Ohio.

Several months ago, *Lifelines/The Software Magazine* (May, 1983) published an article of mine on three packages designed for this purpose which concluded that SUPERVYZ was the most powerful of them but required considerable study by the installer in order to set up the user menus properly. The answer to this problem appears to be ORGANIZER II. With this package, a completely menu-driven system can be developed in a very short time that will, with possibly only one exception, do almost everything you might want to make automatic for your client.

As in all of these systems, they are most valuable when you are using a hard disk with many megabytes of storage and many varied applications. It is only on a hard disk that you could have, on line at all times, word-processing programs and letters, accounting software, a database and database management system, programming software, disk utilities, etc. This presents a confusing set of choices for a user who is only interested in running his business in the most efficient way. The operation of the entire system can be made transparent to the beginner through an efficient front-end menu-driven system.

You can set up menus, menus within menus and pass command line parameters to your command programs. When a command program has completed its run, control returns to a very nicely laid out main menu for further action. The menus, and programs they drive, can be set up easily with your own text editor. In addition, simple menu changes can be made with a built-in menu editor. Providing on board help for your menu entries is extremely easy with no screen formatting required. It's built in.

There is also a screen display language to permit very elaborate demonstration displays and explanatory text.

ORGANIZER II is written in CB80, the Digital Research CBASIC compiler, and is reasonably fast. The package is currently being rewritten in C and will be compiled for the IBM-PC as well as other machines from the C source.

Installation

The first item on the installation menu permits the installation of the user's company name. BE CAREFUL WITH THIS ONE. Once installed it cannot be modified and, unless I am mistaken, if not installed, ORGANIZER II will not work. The name is installed cryptographically and cannot be located in any manner I was able

to discover. This is an attempt by The Information People to limit piracy of their software. It is their hope that a business user would be embarrassed operating software that comes up with the name of some other company and establishes him as a cheat. It is not a new concept but, hopefully, has value. I know that I would not like to install this copy for a customer of mine and have " *Lifelines/The Software Magazine* — Evaluation Copy" displayed on his terminal. There is great controversy over software piracy but, I believe, little disagreement with the concept that, if you're going to make money with a piece of software, you should have paid the author his due.

Installation is very straightforward if you have one of the terminals in the list. Merely select it and you are almost finished. You must also let the system know if you are installing for CP/M or TurboDos. Even though TurboDos claims to run most CP/M software, there are several calls in CB80 programs that must be changed. This feature makes those changes.

The operation of the entire system can be made transparent to the beginner through an efficient front-end menu-driven system.

If your terminal is not listed, you can install the correct codes through the installation procedure once you get these codes from your terminal manual. My Visual 50 was not one of those on the menu of the first version I got. This proved to be no problem and I installed it with no difficulty. Through a question and answer session, you must install the ASCII codes for reverse video attributes, clear to end of line, clear to end of screen, line graphic initialization string, terminal initialization and exit strings, if needed. After this, you may install the control or function keys you want to use to perform the various ORGANIZER functions such as HELP, GET NEXT SCREEN, GET LAST SCREEN, etc. You are also permitted to install your own prompt for any of the controlling functions.

If your terminal has a line graphics mode, you will be required to enter the character needed to produce each type of line or corner symbol.

Usage

The main menu is the first one to appear on the screen. (Sometimes I am in awe at my ability to realize the obvious.) The applications that may be run are located on the left half of the screen. Should you type the chosen control to yield help on any of these applications, this help will appear in a square at the upper half of the right side of the screen. The lower half of the right side displays the controls needed to perform certain operations. You may move the cursor up or down to the application desired. When the cursor is located, you may run this application by pressing <RETURN>, get help with the chosen HELP key or, if there is one, move to the rest of the applications with the selected key, usually the RIGHT ARROW. If a parameter is to be passed to the run program, such as the name of a letter for a word processor, the installer can provide for this entry with a query in the upper right of the screen. Positioning this query (as well as the positioning of the help messages) is taken care by ORGANIZER and cannot be changed by the installer. Nor should I think he would want to change it. The overall layout is well thought out.

Constructing menus

Your text editor should be used to construct menus for your own end user. The menu format is simple. Each line in a menu (.MNU) file can either be a title or subtitle, a blank line or a command line. The blank lines simply make the menu easier to read and you are allowed complete freedom to exercise your own artistry setting up the display. Of course, you must stick to the horizontal size limitations set up by half the screen. The length is up to you. In operation, if the menu should continue past the bottom of the screen, the right arrow key (or whichever key was installed) will allow you to move lower on the menu.

Let us assume that we wish to install a menu line to permit loading the word processor to write a letter to a customer. The command line to enter for this operation might be:

```
Write a letter:?6?2<<WS.COM>>WS
  ((Enter name of letter )); \
      ERA *.BAK; \
      STAT; \
      PAUSE— Be sure to make a
        hard copy for the boss !!
```

Since you have elected to enter a command line, the layout is simple. The "Write a letter," is the description of the operation that will appear in the menu. The end of the description is noted by the appearance of the colon, (':'). The appearance of the ?6? tells the system that, should the operator press the key for HELP (we'll get to this later), help item #6 is the one that must be displayed on the screen.

There are five categories of operations that ORGANIZER II can do and running .COM files is operation 2. That is the meaning of the 2. With ORGANIZER II installed on a hard disk, this next feature will not be needed but it is on a floppy disk system. The use of the double << >> tells ORGANIZER II to check the disk to be sure the enclosed file is on the disk. However, it doesn't do that. It forces a message to be displayed that asks the operator to place the disk with WS.COM in the disk drive. This message is not clear because it appears even if the file searched for is, in actuality, on the disk. I have suggested the wording be changed to "Be sure that WS.COM is on the disk" to avoid a panic by the unknowing operator concerned that the file is not there even though it really is. This search is not required but if the file is not on the disk, the computer reverts to CP/M with no warning. Next is the actual program call. Everything placed between the double parentheses, '((', becomes the prompt for a command file parameter. In this case, it is the name of the letter we wish to write or edit. When ORGANIZER II runs, it will place the line:

```
WS      myletter
```

in the command line. Should you require a series of operations, they may be entered, separated by semi-colons with a final semi-colon at the end. Should you require more than one line for a complex operation, place a backslash, '\', and continue on the

next line. Therefore, in our example line, after completing the letter, the system will erase all .BAK files and run the CP/M utility STAT to let us know how much room is left on the disk.

Your text editor should be used to construct menus for your own end user.

Should you be running a series of operations where you might want to pause between each step, a PAUSE utility is included which permits prompting the user between operations.

The operation codes (the 2 in our example) are:

1 — Invoke the CBASIC interpreter CRUN
2 — COM files
3 — A submenu
4 — Chain an ORGANIZER II overlay (special applications)
5 — Used with MPM (TurboDos ????) to select different user area.

Help

You may write extensive help comments for operation of your menu lines. After installing your menu lines, place ##### on a new line at the end. From here on, you can write to your heart's content and be as verbose as you desire. ORGANIZER II will format your undying prose to fit the upper right square on your screen. Each help entry must be preceded by a ?nn where the nn represents the number you assigned in the above menu to call the help. In our example, this entry might be:

> ?6 WordStar is your wordprocessor. When you enter the name of the letter in the upper right of the screen, WordStar will be loaded into the computer memory with the letter you have named. For specific instructions on WordStar, see the manual.

If the operator tries to invoke help for an operation for which you have not provided any help, a 'No Help Available' message will be displayed.

Disk jockey

Disk Jockey is a utility available with ORGANIZER II (available under CP/M and CP/M look alikes). Invoking this file manager displays your disk directory. The standard disk operations may be performed on each entry or a choice of entries by locating the operation on a menu. The files to be acted upon may be selected or deselected by a one-key action.

You may select the group with the standard CP/M format of wildcard ?'s (the asterisk, '*', is not allowed) as well as Directory/System, Read/Write or Read Only attributes. Systems that use the archive byte can also be selected with this attribute. Once you have the general group displayed, you may select or deselect from within this group with another key. When all files to be acted upon have been narrowed down, you may copy, erase, move, print the directory, print the files, or rename them by placing the cursor on the proper command.

Early in this article, I stated that ORGANIZER II would answer most of the problems of SUPERVYZ with an exception. SUPERVYZ will interact with your user program from within as well as from the command line. For example, the CP/M system copier SYSGEN demands terminal input to specify the source and destination disks of the CP/M system to be copied. SUPERVYZ will set up what might be called a pipeline and redirect input to the program from its own file rather than the keyboard. This is true of input needed for other programs such as BASIC, as well. By this means, almost totally automatic operation can be achieved when the parameters are either fixed or may be entered through the supervising menu program. I would like to see this done by ORGANIZER II but some manner of redirection would have to be established through the operating system.

My manual mentioned a utility called DISPLAY which allows some pretty fancy instructional material to be displayed, movie fashion. I did not find DISPLAY but did find something called ORGDEMO which called an instructional file explaining ORGANIZER II. The menu line that called it was ORGDEMO. I assume that this was simply a name change. If so, the production disks should have the name changed or the manual should be rewritten. It does cause some confusion. This utility presents a file called ORGANIZR.TXT by default. However, use another name in the command line, and DISPLAY will run the named file instead of the default using the display language supplied. Some of the many commands available are:

\ASK	Waits for user to hit a key (or if none, continues automatically after five minutes)
\AT row,col	Positions the cursor and prints text that is on the next line.
\BOO	With no other controls, BOO will display the material 23 lines at a time and delay between screens.
\BOX r,c,w,d	Produces a box with the system line graphics at location r,c and of width w, and depth d.
\FIL file	Displays a file with the name 'file' with all embedded codes.
\SLO	Changes the rate of line-painting until speeded up by the command \FST.

Your inverse video attribute may be turned on and off by a ↑H (that's a caret-H, not control H) before and after the text. This must be done on each line as the attribute does not carry past the carriage return. While the display is moving, you can change the display speed with the up and down arrows. Should you run ORGANIZER II on a DEC Rainbow, you are permitted three other attributes: bold, underscore or blink. It would have been nice to move this on the installation procedure for other terminals. Maybe in future versions?

ORGANIZER II will format your undying prose to fit the upper right square on your screen.

Qualitative Factors

P = supplied by programmer/installer
(0 = low; 7 = high)

Documentation	7
Organized for Learning	7
Organized for Reference	7
Readability	7
Includes all needed information	6
Initial startup	7
Flexibility	7
End User Usefulness	6
Programming Level Needed (0 = none; 7 = much)	
to start	0
advanced work	6
Fulfills Promise	7
Expandability	6
Speed	4
On Board Help (for programmer)	0
On Board Help (for end user)	P
Error recovery:	
From input error	7
From missing files	P
Price	$149

Available from: The Information People
443 Hudson Avenue Newark, Ohio 43055
(614) 349-8644

Feature

TRANSFORM -Structured Translator

by Gregory Knott

Sam: Well, Ned, I got the answers last night for our Las Vegas trip.

Ned: You mean you already wrote that simulation for the game of craps?

S: Yup.

N: Did you use that same logic flow chart that you and I developed the other day?

S: Sure did, and it works great. However, the best thing about the project is that I wrote the program in Microsoft BASIC without using a single GOTO!

N: What do you mean — no GOTOs? You did say BASIC, didn't you? Everybody knows that the GOTO is the mainstay of the BASIC programmer's tool kit. That craps simulation should bave been a short program, but it did have several loops and some IF logic. How could you do it with no GOTOs?

S: I wrote it using totally structured code. . .

N: Fully structured?! With Microsoft BASIC?!

S: Well, that's not entirely correct. I used a pre-processor called the TRANSFORM Structured Translator. All I did was write in a "higher level" BASIC using "structures" like REPEAT - UNTIL and IF - ENDIF. Then I ran the source code through TRANSFORM and it expanded my code into 100% normal version 5.x Microsoft BASIC code which when run under the MBASIC interpreter gave me the answer to our craps simulation.

N: Why on earth would you want to take that extra translation step for such a small program?

S: Just for practice. I've been using this package for a couple of months and it really looks good. Remember when you and I wrote that Job Costing Package for the advertising agency two years ago?

N: I sure do! That was a lot of lines of code. Really, the initial writing wasn't that bad. But when we started system testing, I thought I was going to pull my hair out. That's when I started to lose my patience with the BASIC language. Sure was hard to debug some of those routines written months earlier.

S: That's where TRANSFORM can be a real help. I wish we had had it then. Since you can write modules with one entry and one exit that don't have GOTOs flopping all about, it will be easier to maintain large programs. But

don't forget how much easier it is to also write structured . . .

N: Hold it right there. You know as well as I that I'm a structured code bigot so don't try to sell me on structured programming — tell me more about TRANSFORM. For example: what kind of structures does it have?

S: TRANSFORM has been modeled primarily after Pascal and it appears to contain most of those structures. It has a REPEAT-UNTIL structure that looks like this:

```
REPEAT
  . . .
  block of statements
  . . .
UNTIL (condition)
```

This structure is similar to the MBASIC WHILE-WEND except the loop is executed at least once. The condition test is made after the block of statements instead of before as in the WHILE-WEND.

WHILE-WEND is also allowed in TRANSFORM, however, the pre-processor doesn't modify it; that is, it works the same as it does in MBASIC.

The IF-ENDIF structure is an expansion of the standard MBASIC IF statement. The translator allows large blocks of statements to be used after THEN, ELSEIF, and ELSE; that is, it is not necessary to require all code to be on one line. It looks like this:

```
IF (condition1) [THEN]
  . . .
  block of statements
  . . .
[ELSEIF (condition2) [THEN]
  . . .
  block of statements]
  . . .
[ELSE
  . . .
  block of statements]
  . . .
ENDIF
```

TRANSFORM also has an improved ON-GOTO-ENDGOTO structure that looks like this:

```
ON (integer expression) GOTO @LABEL1
    [,@LABEL2 [,. . . [,@LABELn]]]
    . . .
    @LABEL1
      block of statements
      . . .
    [@LABEL2
      . . .
    block of statements]
      . . .
    [@LABELn
      . . .
      block of statements]
      . . .
    ENDGOTO
```

The operation of this command differs from MBASIC because the translator automatically inserts a jump to the line containing the ENDGOTO after the statements in each block. In this way, one and only one block of statements will be executed per pass through the ON-GOTO.

N: Wait a minute. I'm following you up to this point but now you're starting to talk about labels. You mean line numbers, right?

S: No, no, no. I got ahead of myself. One of the things I like is that with TRANSFORM you don't need line numbers. You use labels instead. Let's look at my craps program as I wrote it in "TRANSFORMese." Even though it isn't well commented you don't really have any trouble following the program.

```
DEFINT A-Z
FIRSTWINS = 0
FIRSTLOSS = 0
POINTWINS = 0
POINTLOSS = 0
FOR GAMECOUNT = 1 TO 10000
  GOSUB @ROLL__THE__DICE
  IF DICE.TOTAL = 7 OR DICE.TOTAL = 11 THEN
    FIRSTWINS = FIRSTWINS + 1
  ELSEIF DICE.TOTAL < 4 OR DICE.TOTAL = 12 THEN
    FIRSTLOSS = FIRSTLOSS + 1
  ELSE
    POINT = DICE.TOTAL
    REPEAT
      GOSUB @ROLL__THE__DICE
    UNTIL DICE.TOTAL = 7 OR DICE.TOTAL = POINT
    IF DICE.TOTAL = POINT THEN
      POINTWINS = POINTWINS + 1
    ELSE
      POINTLOSS = POINTLOSS + 1
    ENDIF
  ENDIF
NEXT GAMECOUNT
'
PRINT "FIRST ROUND WINS: ", FIRSTWINS
PRINT "FIRST ROUND LOSES:", FIRSTLOSS
PRINT "POINT WINS:       ", POINTWINS
PRINT "POINT LOSS:       ",POINTLOSS
PRINT "TOTAL GAMES:      ", GAMECOUNT - 1
STOP
'
@ROLL__THE__DICE
DIE1 = INT(6 * RND + 1)
DIE2 = INT(6 * RND + 1)
DICE.TOTAL = DIE1 + DIE2
RETURN
'
END
```

I initialize my counters, then go into the program's main loop where I have decided to play 10,000 games. After rolling the dice — if we get a 7 or 11 we won on the first roll. If a 2, 3, or 12 came up we lost. Otherwise, whatever we rolled became the point and we continued rolling the dice until either a 7 or our point was rolled again. If we rolled our point we won the game; otherwise, we lost. After 10,000 games I displayed the results.

N: Your indentation scheme made it easy to follow and because of labels instead of the numbers when you jump to a subroutine, I really didn't have to go look at that code. The name you gave that label, ROLL__THE__DICE, told the story. Much better than GOSUB 11000.

S: After I ran my source code through the TRANSFORM Structured Translator this is what I got:

```
1000  DEFINT A-Z
1010  FIRSTWINS = 0
1020  FIRSTLOSS = 0
1030  POINTWINS = 0
```

►

```
1040  POINTLOSS = 0
1050  FOR GAMECOUNT = 1 TO 10000
1060  GOSUB 1350
1070  IF NOT (DICE.TOTAL = 7 OR DICE.TOTAL = 11 ) THEN 1100
1080  FIRSTWINS = FIRSTWINS + 1
1090  GOTO 1250
1100  IF NOT (DICE.TOTAL < 4 OR DICE.TOTAL = 12 ) THEN 1130
1110  FIRSTLOSS = FIRSTLOSS + 1
1120  GOTO 1240
1130  ' ELSE
1140  POINT = DICE.TOTAL
1150  ' REPEAT
1160  GOSUB 1350
1170  IF NOT (DICE.TOTAL = 7 OR DICE.TOTAL = POINT) THEN 1150
1180  IF NOT (DICE.TOTAL = POINT ) THEN 1210
1190  POINTWINS = POINTWINS + 1
1200  GOTO 1230
1210  ' ELSE
1220  POINTLOSS = POINTLOSS + 1
1230  ' ENDIF
1240  ' ENDIF
1250  ' ENDIF
1260  NEXT GAMECOUNT
1270  '
1280  PRINT "FIRST ROUND WINS: ", FIRSTWINS
1290  PRINT "FIRST ROUND LOSES:", FIRSTLOSS
1300  PRINT "POINT WINS:       ", POINTWINS
1310  PRINT "POINT LOSS:       ", POINTLOSS
1320  PRINT "TOTAL GAMES:      ", GAMECOUNT - 1
1330  STOP
1340  '
1350  ' ROLL_THE_DICE
1360  DIE1 = INT(6 * RND + 1)
1370  DIE2 = INT(6 * RND + 1)
1380  DICE.TOTAL = DIE1 + DIE2
1390  RETURN
1400  '
1410  END
```

N: Oh, yes. That looks more familiar. The GOTOs and line numbers that MBASIC needs have been generated by TRANSFORM.

S: Yep. All the structures used are expanded so that MBASIC will understand what is going on. TRANSFORM also has an INCLUDE facility which allows me to bring common blocks of code and substitute global (program) variables for the local (INCLUDEd block of statements) variables. You can even nest INCLUDEd files to a total level of five INCLUDEs, each with its own parameter substitution.

A switch allows temporarily turning off the translator so particular sequences of MBASIC code will not be expanded. It allows you to specify the beginning line number and the increment of the converted source file, print the destination program and a user label table.

After the translation, an error listing prints out. For the craps program it looked like this:

```
0    prepcompilation error(s) detected.
8    labels used: space for 342 remaining.

No. IFs:        2  No. ENDIFs:     2
No. WHILEs:     0  No. WENDs:      0
No. REPEATs:    1  No. UNTILs:     1
No. FORs:       1  No. NEXTs:      1
No. ONs:        0  No. ENDGOTOs:   0
```

N: This program sounds great, but doesn't it add an extra step to the programming process?

S: Right you are. One of the nice things about the MBASIC interpreter is that once a program change is made you can simply type RUN and see the results. With TRANSFORM, making a change in the program requires most of these steps:

- bring up an editor or word processor
- load the program you are writing (pre-processed format)
- make your program changes
- save the program
- exit the word processor and bring up TRANSFORM
- respond to TRANSFORM's prompts with the above program name
- after TRANSFORM writes the post-processed program to disk, exit TRANSFORM and bring up the BASIC interpreter
- load the TRANSFORMed program and commence testing.

However, if you write a lot of BASIC code and possibly maintain that code months later, the niceties described earlier (e.g., structure, labels, includes) would probably far outweigh the added inconvenience. Besides, under the interpreter you could fix a logic bug, run it, fix the next one, etc., before jumping back to the word processor. This way you could fix many bugs at once — you can't do that in other "high-level" languages. I think this package was made for the serious MBASIC programmer. Those who dabble with MBASIC now and then wouldn't have much use for it but those programmers who put out many lines of code may find this just the ticket.

N: Another individual who might want to buy it would be that person wanting to learn structured programming, who already had an investment in MBASIC, and didn't want to put out heavy money for a full fledged high-level, compiled language. By the way, how much does this package set one back anyway?

S: The company sells this product in three different packages:

First, there is the Standard package which consists of the TRANSFORM Structured Translator, a tutorial, and a few utilities, the best of which is a "Pretty-Printing" BASIC Page Print Utility. (Much better than LLIST).

Second, there is the expert package containing TRANSFORM, an optimizing option for the BASIC Compiler, a Page Print utility and substituted for the tutorial information, a really unique utility. This utility is a REL file disassembler which produces a PRN or MAC file from a relocatable load module. Although I have not played with it, it appears to be quite powerful. I envision it being used for changing some of the library routines issued with some of the compiled languages. I also believe that this disassembler might be the only product of its type on the market and may be worth the price of the product alone. There are also some other utilities which all add up to tools to create very elegant code.

Third, there is a beginning package which contains a very extensive tutorial that takes a person from a very low (non-existent) understanding of structured programming and builds knowledge and confidence in the techniques involved by developing its material around sample programs.

N: Three different ways to purchase TRANSFORM. The Standard, Expert, and Beginning packages. You still didn't say how much it costs.

S: You're always in such a hurry. The price of each package is $39.95. If more than one package is purchased, add $16.95 for each additional one. The product can be purchased from MasterComputing Inc., P.O. Box 17442, Greenville, SC 29606, Phone (803) 244-8174.

N: Did you find any bugs in the program? Is there anything you would have it do differently?

S: No bugs. By calling the company I found out that after issuing over 100 copies of the product they have yet to receive any user reports. They themselves found two minor bugs which have been fixed in their latest release. The product is actually written in itself.

In using the product, I thought that any seasoned programmer would be ecstatic over the documentation and with a couple of close readings of the main manual and a look through the tutorial information, most first timers would have little trouble.

Since I view this more as a production tool, there is one area I would like to see streamlined. Upon entering TRANSFORM you are led down a prompting path which is very easy and straightforward but typically you end up defaulting through most of the prompts. There is no facility for going around these prompts. I think if one were able to enter A>TRANSFORM CRAPS on the initial start-up line it would be faster and easier. If, as your application were nearing its final stages, you wanted to do something special with the prompt responses by leaving off the file-name the program could revert to prompting.

Also, when entering the final pass of a fully tested program, the program should allow "scrunching" of the MBASIC code, putting many commands on one line, removing the remarks, etc. This should help to speed up execution of the finished program. This optimizing option would take effect toward the end of the development cycle, thus allowing easier debugging during the early stages of development.

N: Boy, you have me convinced. I think I'll add it to my library of software programs. By the way, care to discuss how we're going to do at the craps table in Vegas?

S: Sure. Here are the results:

```
FIRST ROUND WINS: .2144
FIRST ROUND LOSES: .1091
POINT WINS: . . . . . .2760
POINT LOSSES: . . . . . .4005
TOTAL GAMES: . . . . .10000
```

I'm not sure craps is our game. Maybe we should write a different simulation for, say, blackjack.

N: OK. But let's do it on your computer. I'd like to get a closer look at TRANSFORM.

S: Let's go. By the way, did I tell you that when TRANSFORM is printing your results it also prints a number at the left which tells how many levels that statement is nested inside another?

And also. . . . . ■

SOFTWARE NOTES

T-Maker Tips

T/Maker can work with any printer as is. It can do bold and underscore and overprint. However, if you want to use any other special features of your printer, you have to set up a PRINT.UTL file.

T/Maker III can control the special features of your printer through the use of a PRINT.UTL file. For example, you can use the condensed type font, or the enlarged, or the italics of your dot matrix printer. If you have a letter-quality printer, you can use the special character sets or special characters.

After you create a PRINT.UTL file and make some minor changes using TMODIFY, you can use all the functions of your specific printer. Conceptually, this is how it works: during the PRINT process, when T/Maker encounters a special character (one which you have defined), it will send out a specific sequence of control characters which turn on/off one of your printer's special features (condense, enlarge, bold, etc.). For example, let's say that you want the ↑ character to mean the start of condensed printing and the ~ character to mean the end of condensed printing. So, you would use them as such:

```
This is normal spacing—↑this is condensed~
—this is normal again.
```

You designate which special character you want to signify the beginning and end of any special feature. These characters are typed directly into the file just like any other character. When the PRINT command is given, T/Maker starts printing the file. When it encounters a special character (in this example: ↑ or ~), it sends out the specific control sequence to the printer.

With respect to the special characters that you choose to indicate the start of specific control sequences, you want these characters to be ones you don't normally use. As in the previous example, you could use ↑ and ~, but you may want to use them normally and not necessarily have them send out a control sequence. Therefore, T/Maker allows you to set up an alternate set of characters that are used only for designating the beginning and ending of the printer control sequences. Every character has an associated ASCII code (eg., A = 65, a = 97) and the alternate character set starts with character codes greater than 128.

To create this alternate character set you must first designate a "high bit" character. When typed, this character will not appear on the screen but will add 128 to the next typed character. Say, for example, the high bit character was the @ sign (@ on CP/M versions and ` on MS-DOS versions). If you typed @ and then "A," the result would be a single character with the ASCII value 128 + 97 = 225.

Now that you have the background, there are two main things that you have to do to get rolling. First, create a PRINT.UTL table using the T/Maker editor. Create a file similar to the following: ►

(Continued on page 16)

Review

Benchmark 3.0

by Van Court Hare

Typing pool people do two types of work: first, they must create documents quickly in draft form; then, more deliberately, they edit, style, and correct the final manuscript to incorporate needed revisions.

When worker selection correctly matches typists to one or the other of these two — extremely different — tasks, both overall typing efficiency and worker morale improve. One person is seldom best at both high speed and precision work.

A two-phase wordprocessor

Benchmark Version 3.0 (Metasoft Corporation, 6509 West Frye Road, Chandler, AZ 85224, 800-621-1908) is specifically designed for such a split division of labor. Consequently, Benchmark is more an office product than one designed for the individual: its structure, its menu driven options, and its special features all point to the automation of repetitive, predictable activities — not to highly volatile text creation and alteration done interchangeably.

Benchmark's structure

Benchmark's structure follows the traditional author/publisher model and completely separates document creation (Benchmark's 'Insert Mode') from document editing and styling (Benchmark's 'Control Mode'). Although a single Benchmark user can move from text creation to the text edit mode (and back again, if necessary), for best results this change of mode should be limited to large blocks of text.

Menus totally drive the whole Benchmark system, a design which forces all users to conform to the same office standards — to the delight of supervisors!

There is, in fact, no way around Benchmark's initial menus, which demand complete title, author, typist, and revision number identification each time you create or revise a document. Benchmark automatically increments document revision numbers each time a new revision is requested; and, should you forget which documents or which revisions are on file, Benchmark will bring up the whole completely identified directory on its screen with one keystroke. Clearly, accurate document ID tags are a must in any busy office, so you don't mind that. (My Benchmark review copy, Version 3.0L, which is customized for a NEC Advanced Personal Computer, also automatically stamps each revision file with a date and time entry from the APC's internal real time calendar/clock, a nice touch.)

Next, you move to Benchmark's Insert or Control Mode, depending on your need.

Benchmark is more an office product than one designed for the individual.

During actual document creation or revision, a ubiquitous menu sequence guides the Benchmark user through the options and procedures of this extensive wordprocessing system. For example, even such details as Page Format and Print Style are changed by menu selection. The menu approach is good for clerical instruction, jogs the memory of more experienced users, and promotes accurate work — even though you pay some penalty in speed for this constant prompting.

Finally, when a document creation or revision session is complete, you strike a function key defined as FINISH, and may retain or delete prior revisions of the current document at that time. Like a persistent spouse, Benchmark's prompt reminds you to take out the garbage. You are now returned to the main Benchmark menu, from which you can start again to create or revise. From the main Benchmark menu, you may also selectively (or batch) delete or print named files, perform several additional procedures — such as file conversions to and from ASCII format — or exit the wordprocessor.

Benchmark's document filing capability is greatly enhanced over normal CP/M systems by a special design feature worth noting: Metasoft places its own supervisory monitor program between the normal operating system and Benchmark itself. This custom monitor redefines and expands the usual OS file management procedures and permits expanded directories, indexed document segments *within* named files for easy boilerplate maintenance (as illustrated subsequently), and full control of up to seven different "Storage Units" for convenient file grouping.

Because Benchmark is usually set up to autoload all necessary files when you insert the system disk, the first screen you see is an initial choice menu. No operating system complexities ever appear. No computer knowledge is required to use Benchmark effectively. This "turnkey" approach, of course, is ideal for clerical personnel and is part of Benchmark's overall philosophy.

Moreover, Benchmark has been carefully integrated with the keyboard function keys for many specific computing systems, and drivers for popular printers are provided. An on-line installation menu permits default or temporary system alteration. The net result is a clean package which will have great appeal to those who are not computer specialists.

Prices for Benchmark vary somewhat ($350-575), depending on the amount of system integration, the license agreement provided, and the vendor. A buyer should make certain to get the specific Benchmark version tailored to his or her machine, and accept no substitute. On some occasions, manufacturers have offered Benchmark with their hardware on a promotional basis. You may wish to check that out before you buy because Benchmark is best

viewed in a given hardware/software setting.

An ideal hardware/software marriage

To illustrate, my NEC Benchmark is completely integrated with APC hardware, specifically the 44 function keys on the APC keyboard. An overlay template in color, included with NEC's Benchmark, identifies each function key clearly. These keys generate Benchmark's underlying control key sequences, which alternatively may be typed directly. Owners of NEC's APC Model HO-3 color unit will also have color support to highlight text alterations. Reverse video and intensity shades appear on the monochrome HO-1 and HO-2.

NEC also offers an interesting "personal" license for this product: you are licensed to use your copy of Benchmark on any NEC APC machine you are authorized to operate. For example, if you have one APC at home and another at the office, your Benchmark can legally go back and forth with you. After all, NEC sells hardware first, and structures their deals accordingly. As usual, making copies for other persons is prohibited.

The two megabyte capacity of the APC's 8" DS/DD disks and the extremely high resolution of the APC screen (1024 x 1024 in both monochrome and color), make the APC hardware and Benchmark a truly outstanding wordprocessing combination. The NEC 7220 video generator chip and high bandwidth monitor are responsible for the commercial quality of this setup; you can type all day and feel no eyestrain!

I gave my fifteen-year-old son NEC's Benchmark manual, showed him how to insert the disks properly into an HO-2, and turned him loose on the system just described. (It was his first wordprocessing effort.) Within the hour, Vano had produced printed output — notes for his Botany class. He loves Benchmark, and doesn't want to learn anything else — considerable testimonial to NEC's APC implementation of the Benchmark program! As a first user, Vano appreciates Benchmark's menu guidance. And, I can rest assured that my teenager will not damage the battleship strong APC hardware: under the APC's light beige molded case, the guts are framed in eighth-inch steel.

The only NEC APC hardware blemish I found which affects Benchmark is the keyboard placement of the "Graph 1" and "Graph 2" keys, which shift the keyboard to display Greek and math symbols, or other graphics characters. Inadvertent depression of either "Graph" key will lock up Benchmark unexpectedly. (Unfortunately, Benchmark does not recognize these special CRT symbols.) Recovery is easy: just unlatch the Graph key(s) and continue. Users soon learn not to latch the Graphics keys by mistake; nevertheless, their placement is not ideal.

Metasoft places its own supervisory monitor program between the normal operating system and Benchmark itself.

The big Benchmark feature: Predefined inserts

Benchmark is no kid's toy: it has many special features to serve its intended office-oriented purpose.

First, Benchmark can define and store up to 52 phrases (each up to 2000 characters long); then you can retrieve any selection with a keystroke for fast insertion into a document. These phrases reside on the system disk, but by appropriate creation of additional system disks, one for each typical subject, you can create an unlimited phrase library.

"Indexed" insertion files are even more powerful. This second approach, a special Benchmark feature, allows segments within a given file to be indexed by name. Then, after the indexed file has been prepared, a new document file can be assembled rapidly by reference to the indexed blocks desired. Benchmark's indexed file segments can range from a sentence to paragraphs, or pages. The number of indexed segments you can maintain is limited only by the amount of disk media available.

Suppose a legal office prepares standard paragraphs for wills, as shown in simplified form in Figure 1. In this figure, three stock paragraphs are identified as "segments," each delimited by a pair of left and right brackets [ ]. The segment name (to be indexed) is the first phrase shown in each segment. This name tag starts with DB: (for "define block") and ends with a RETURN, marked in Figure 1, as on the Benchmark screen, by the < symbol.

Figure 1

```
[DB:will 1<
I, [Testator], being of sound mind, declare this to be my last will and testament.<
<
]<
[DB:will 2<
I hereby bequeath to my husband, [Husband], all property I own or possess on the date of my death.<
<
]<
[DB:will 3<
I hereby bequeath to my wife, [Wife], all property I own or possess on the date of my death.<
]
```

You index this Master Document by a procedure selected from the Benchmark main menu. Once this is done, subsequent documents can ask for any paragraph by name. Thus, for a husband, you could create a new file consisting of the line

[will 1][will 3]

Next, from the Benchmark "Control Mode," you "attach" the indexed Master Document to the new document by typing "A" or the equivalent function key. Finally, you strike the "Q" (or "Quick Merge" function) key. The new document will be assembled rapidly from the indexed file without further operator action.

Such an assembly permits "nesting," so the [Testator] and [Wife] inserts still appear. A second application of Quick Merge permits manual fills as needed, and the inserts will be justified and aligned according to the Page Format in current use. (For extensions of this procedure, you can use Metasoft's optional Mail List program.)

►

I have used Benchmark's indexed block procedure, just described, to create student exams quickly. From a data bank of serially indexed questions, I simply select a set of desired questions by number and assemble them. The proper question sequence numbers for a new quiz can be filled in (like [Wife] in the will example) as a final manual procedure.

To set up many of Benchmark's other block operations (delete, move, exchange, heading/footing definition, and so on), you depress the desired function (or corresponding keyboard) key to mark the block start. The marked spot appears in reverse video (or color), and the keyboard is temporarily redefined.

Now when you strike any key, the marked block will expand to reach the character struck. In this way, a space will mark one word, a RETURN will mark to end of line, a period to end of sentence, and so forth. The cursor-right key will mark one character at a time. (If you go too far, the ESC key will undo the marking and restart you.) The Benchmark marking process is much faster than it seems from this description, and you rapidly get used to it.

"Indexed" insertion files are even more powerful.

Finally, to perform the desired block operation, you strike one of the function keys defined as EXECUTE, and the deed is done. Once the EXECUTE key has been depressed, say for a delete operation, no automatic recovery is possible.

The standard "Function-Do-Execute" sequence

In general, most Benchmark keyboard entries follow a sequence like that just described. Each specific operation, which shifts a keyboard mode, is terminated by striking the EXECUTE function key or its equivalent. Because the EXECUTE key is the one used most heavily, four of the NEC APC function keys — at the very center of the function key strip and marked in bright blue on the Benchmark keyboard template — are devoted to this operation.

To illustrate further, if you are in "Control Mode" and wish to insert text, you strike "I" or the Insert Function Key. The screen and keyboard shift to "Insert Mode" and you may now enter text at will. When done, you strike EXECUTE, and you are back in "Control Mode." The text automatically adjusts to the current Page Format in use.

You must follow this same three-step sequence to overtype, or to perform other editing steps. Indeed, if you wish to insert or overtype or delete only one character when revising a document, you must still strike a minimum of three keys!

I found this three-step constraint — which slows down single character corrections markedly — to be Benchmark's greatest shortcoming and source of frustration (compare to WordStar's single stroke control key inserts and deletes). In Benchmark's "Insert Mode" the backspace key does delete the single character to the left, but this is too limited a correction feature for creative authors, and a constant flip between Insert and Control Mode is totally unfeasible. (That is why Benchmark becomes most useful only in the split document preparation process initially suggested at the start of this article.)

Fortunately, Benchmark's three-step sequence seems quite natural for operations on large blocks of text.

What you see is not exactly what you get

Benchmark text always appears on the screen single spaced. You use the Page Format and Print Style menus (at any point) from Benchmark's Control Mode to adjust page layout and character changes (underline, bold, shadow, superscript, subscript, and such).

Format and style changes are marked on the screen, as are correctly computed page breaks; but line spacing never changes. Most of the obvious format features of a document (justified lines, indents, and the like) appear on the screen as they occur. However, to see the complete Page Format status at any point in the text, you must look at the current Page Format Menu where the text mark "# - Page Format" appears. The latter format change marker automatically appears on the screen after each format change you make.

Similarly, the circumflex symbol "↑" appears on the screen each time you initiate a Print Style change. This mark clearly shows where you requested styling; but, alas, to make that change or to know what was previously done, you must call up the Print Style Menu. (Compare to WordStar's ↑P key sequence and the resulting style marks, like ↑B or ↑S.)

The Benchmark Page Format and Print Style Menus permit unlimited print control and are a good reminder of what can be done — which is more than considerable. The menu approach also promotes precision in style entry. Nevertheless, the application of the Page Format and Print Style Menus is a somewhat tedious and slow business, and no job for a beginner. Yes, yet another reason to divide up the typing pool by personality type.

Other features

Space limits full description of Benchmark's other options, some of which are unique in a single word-processing package. Briefly, in addition to the extensive boilerplate options already mentioned, Benchmark provides:

(1) column editing ability, so you can generate columnar text and financial tables, properly aligned;

(2) a calculator mode for simple row and column arithmetic;

(3) a limited business graphics capability, to draw horizontal and vertical lines and boxes;

(4) extensive printing facilities, with page numbering, headings, footings, footnotes, left and right control, full character and page formatting, multiple-document batch printing, and so on;

(5) plus utilities for file conversion from Benchmark to CP/M-86 ASCII files, and vice versa. (The styling marks of non-Benchmark wordprocessors [usually carriage returns] must be cleaned up on the Benchmark screen; conversion from Benchmark to ASCII strips Benchmark's control codes.)

The NEC Benchmark manual is typeset and printed in two colors with full screen illustrations for each feature. A full index and chapter divider tabs make reference easy, and the text is cookbook clear. An extensive reference card also comes with the package.

and the text is cookbook clear. An extensive reference card also comes with the package.

Metasoft supplies Mail List Merge, Spelling Check, and Telecommunications options for Benchmark. All of these programs can be run from the main monitor menu which controls the wordprocessor. Although none of these extensions has been reviewed to date, if they are of Benchmark's quality, they are superior products.

Some words of caution

If you use Benchmark, take care to note whether you are in the Insert or the Control Mode before you strike a key! The Insert/Control split permits Benchmark to use single keystrokes for most control operations, and in this sense the Benchmark "Control Mode" keyboard is truly "hot" — nearly every key will do something which is not in effect, or which acts differently in "Insert Mode."

Former users of other wordprocessors must beware: attempts to use WordStar (or other) control key sequences in *any* Benchmark mode or menu will promote disaster, or at least frustration. For this reason, my daughter, Bailey, sticks to WordStar for her college term papers — not only because she learned WordStar first, but also because she sees Benchmark as too rigid for her creative efforts.

If you use Benchmark, take care to note whether you are in the Insert or the Control mode before you strike a key!

Benchmark does a lot of automatic disk backup, and the totally menu-driven design often requires disk action, too. Accordingly, if you try to create and correct simultaneously, Benchmark *is* sluggish and awkward. But just wait until the time you need lots of stock paragraphs, statistical inserts, or heavy-duty printing! Benchmark lets you menu-select multiple documents and print them all at once, in any order — without extra submit files or other to do.

Remember, Benchmark is primarily an office-oriented product!

Conclusion

Benchmark, King of the Microprocessor Supported Clerical Office, has a well chosen name. Its design supports known typing pool demands and high-turnover clerical personnel characteristics — and does so admirably. Program developers and creative authors need not apply. ■

Mea Culpa

by Bruce H. Hunter

I received a very nice letter from Jacob L. Warner, of Greenbelt, Maryland, who requested more program examples in the article series. "I realize that your space in *Lifelines/The Software Magazine* is limited but, in several cases, more examples are badly needed. One of the things that makes PL/I difficult to learn is that one sees so few programs in the language in magazines, etc. In my view an example is worth a thousand words . . . ." This is aptly put, and in response to this request, I created a program for the chapter on pointers in this issue. There will be more program examples from now on.

Mr. Warner has some things to add to my description of the verify() function in Installment No. 3. "I don't think you did justice to VERIFY, as you did not mention its primary use as a statement to verify that the input is one of several proper characters. If VERIFY (choice, 'YN') = 0, for instance, then a choice in the proper range Y,N has been made. I was surprised that you did not make use of this statement in Installment No. 4 when you were discussing bullet-proofing techniques. On the other hand, I had not thought of using VERIFY as you used it in your 'convert' function . . . ." Mr. Warner makes a good point about the verify() function, and I'm glad for his input.

He goes on to say, "In installment No. 3, your description of TRANSLATE is amusing, but it really is a useful statement. I enclose my function for using it to transform to upper or lower case. The nice thing is no test has to be made to see that the input string characters are in the proper range — if not, they are simply reproduced, not changed." As an author I am often tempted to go for an occasional laugh at the expense of an accurate description, so I'm including Mr. Warner's program for the benefit of others who may not have foreseen the usefulness of the translate() function:

```
type casetest.pli
casetest: proc options(main);
      dcl
      k char(254) var;
      %replace
      true by '1'b;
      do while(true);
      put skip list('Input up to 254 characters: ');
      get edit(k) (a);
      put skip list(cap(k));
      put skip list(uncap(k));
      end;
cap: proc(stringin) returns(char(254) var);
      dcl
      (stringin,stringout) char(254) var,
      (upper,lower) char(26);
      upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      lower = 'abcdefghijklmnopqrstuvwxyz';
      stringout = translate(stringin,upper,lower);
      return (stringout);
end cap;
uncap: proc(stringin) returns(char(254) var);
      dcl
      (stringin,stringout) char(254) var,
      (upper,lower) char(26);
      upper = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      lower = 'abcdefghijklmnopqrstuvwxyz';
      stringout = translate(stringin,lower,upper);
      return(stringout);
end uncap;
end casetest;
```

Thank you, Jake, for your letter and your input! I welcome all comments from readers. PL/I can get "heavy," as in this month's article on pointers, and any comments that correct, enhance or clarify can only be helpful to those learning the language. So keep those letters coming!

Hunter Structured Software
1020 South Jennifer, Glendora, CA 91740

(Continued from page 11)

File PRINT.UTL
For the Epson Printer

Char Found	Character Sequence sent to printer	End sym	Remarks	To Get, type
210	27 64	128	Reset	` R
197	14	128	Double Width-On	` E
206	20	128	Double Width-Off	` N
195	15	128	Condense-On	` C
227	18	128	Condense-Off	` c
194	27 69	128	Bold (emphasize)	` B
226	27 70	128	Bold-Off	` b
175	27 =S 0	128	Superscript-On	` /
191	27 =S 1	128	Subscript-On	` ?
173	27 =T	128	Super/Sub-Off	` –
196	27 =G	128	Double Print-On	` D
228	27 =H	128	Double Print-Off	` d
213	27 45 1	128	Underline-On	` U
245	27 45 0	128	Underline-Off	` u
201	27 52	128	Italics-On	` I
233	27 53	128	Italics-Off	` i
208	27 56	128	Paper End-Off	` P
209	27 57	128	Paper End-On	` Q
204	27 79	128	Perf Skip-Off	` L
198	27 78	128	Perf Skip-On	` F
211	27 65 =10	128	10/72 Line Space	` S
177	27 =K 6 0 255 255 255 255 255 255	128	Dark Block	` 1
178	27 =K 6 0 255 153 153 153 153 255	128	Barred Block	` 2
179	27 =K 6 0 255 129 153 153 129 255	128	Dotted Block	` 3
180	27 =K 6 0 255 129 129 129 129 255	128	Hollow Block	` 4
181	27 =K 6 0 0 60 60 60 60 0	128	Bullet	` 5
182	27 =K 6 0 28 14 7 30 120 240	128	Check Mark	` 6

A couple of points to note:

Char Found: is the alternate character that tells T/Maker to send out the associated control sequence to the printer.

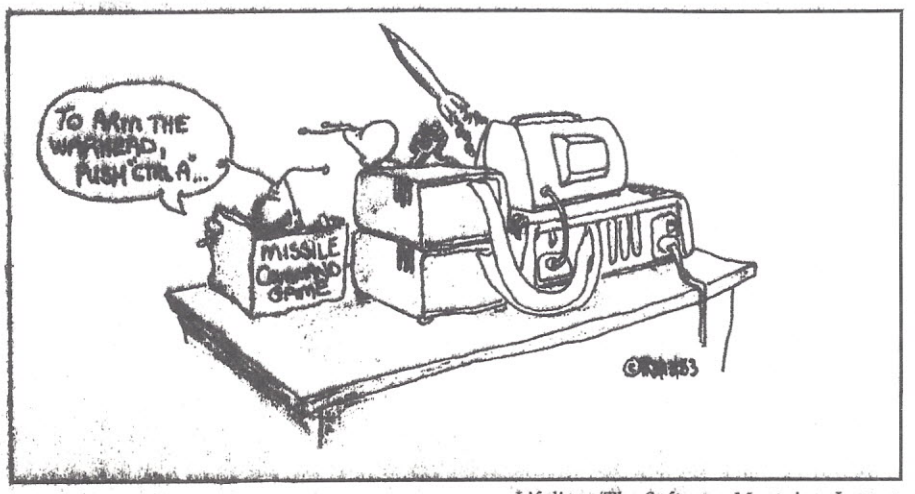
Char Sent: is the control sequence to turn on/off the special feature of the printer (eg., condense, enlarge).

End Sym: This number is the same number that you set during TMODIFY. It tells T/Maker that the control sequence has ended. This end symbol is not sent out to the printer.

Remarks: These are notes to yourself to remember what the settings are. They are not used by T/Maker.

The second step is to make two minor changes using TMODIFY. Type TMODIFY and select option #3. In response to the question "Should characters be translated when printing?", answer YES. In response to the question "What number should end a translation table line?", answer the same number as you used for END SYM in the PRINT.UTL file. Save the changes and exit TMODIFY.

Now you're ready to go. ■



Feature

CB80 Forum

by John S. Coggeshall

Herewith continues a series of technical notes on the CB80 compiler, by Digital Research. As before, I invite you to participate by contacting me through Lifelines/The Software Magazine *or at:*

151 West School House Lane
Philadelphia, PA 19144

Let's take a quick look at CB80's implementation of relational tests for various data types. All tests of the form IF (condition) THEN (whatever) use the same structure:

1) get argument(s), do any calculations required
2) do something that will set a suitable CPU flag if condition is not met
3) conditional jump to 'else' on the flag
4) (whatever)
5) 'else:' label and code

If the ELSE option is used, Step 4 ends with a jump past the code of Step 5, but this does not concern us here. The focus will be on CB80's ways of implementing Step 2. Thus, for example, IF A$<B$ THEN RETURN generates

```
1.      LD      HL,(A$)
        EX      DE,HL
        LD      HL,(B$)
2.      CALL    ?CSMM           Compare Strings
                                (DE) with (HL)
3.      JP      NC,else
4.      RET
5.      else:
```

The library routine ?CSMM, like those for comparing floating-point values, returns with various CPU flags set: the Zero Flag if the values are equal, the Carry Flag if the first is less than the second, and the Sign Flag if the first is greater.

The only chance to beat the system here is in determining whether a string is null: IF LEN(A$) is four bytes shorter than IF A$>"" (counting the two hard data bytes) and certainly faster. Analogously, the only remotely interesting special case involving floats is comparison with zero: IF R1 is a byte shorter and probably faster than IF R1<> ZERO. Both shortcuts involve conversion to an integer.

Things get more lively when it comes to integer tests. DRI has gone out of its way to test them with style. There are no library calls; all testing is done in-line, resulting in possibly bulkier but much faster code. The general strategy is to transpose the condition logically, by the algebra of inequalities, to a form in which something is implicitly compared to zero. For example, IF I%=J% is logically transposed to its equivalent, IF I%–J%=0. (Here and in what follows, I% and J% refer to integer variables, constants, or arithmetic expressions, but not to logical or relational expressions. A distinction between relational expressions and relational tests will develop.) Typically, just enough of the subtraction is performed to set the appropriate CPU flag:

```
1.      ( get I% and J% into DE and HL )
2.      LD      A,E             Partial DE-HL
        SUB     L               (high byte of result
                                not moved to H)
        LD      L,A
        LD      A,D
        SBC     A,H
        OR      L               Does I%–J%=0?
3.      JP      NZ,else         ( or JP Z,else for <> )
```

Note that if you try to force the transposition by coding IF I%–J%=0, CB80 will take you seriously and generate two more bytes, to get the full result into HL and then back to the A register to check its value. So let CB80 handle such refinements. The same applies to the complement: coding IF I%<>J% as IF I%–J% will gain you nothing.

Now suppose J% is an integer constant; IF I%=k, with k between 8 and –8, generates the sort of trick we saw last month:

```
1.      ( get I% into HL )
        DEC     HL              (k times, if k is
                                >=0)
or      INC     HL              (k times, if k is <0)
2.      LD      A,H
        OR      L
3.      JP      NZ,else         ( or JP Z,else for <> )
```

This is the same logical transposition as above, carried out in fewer bytes. Actually, I confess puzzlement here: the code size breaks even at k=7 and the speed at k=5; so for k=8, CB80 surprisingly generates slightly more and slower code. Maybe DRI just got carried away. The same trick is used for small k with the '<' and '>=' operators, but not with '>' and '<='.

All other integer tests generate the partial subtraction (abbreviated still further) and use the Sign Flag. The various forms are summarized in Table 1. Note that, for example, IF I%>0 will make four more bytes than IF I%>=1, which in the case of integers is equivalent.

As VanNatta has pointed out [1], it is worthwhile in terms of speed to nest conditionals rather than compound them. Statements like IF A%=B% AND C%=0 are not only slower than IF A%=B% THEN IF C%=0, but they also require significantly more code (31 versus 24 bytes in this example).

True and false: LOWCASE$ revisited

Now, I hate to belabor a nice routine like VanNatta's Lowcase$ ([1] and Randy Kimbro's letter, May 1983), but Mr. Kimbro raises an interesting issue, and this seems a good place to discuss it. He speaks as though the 'IF' condition must evaluate to a minus one in order for the 'true' branch to be taken.

When the condition is an integer expression with no relational operators, the IF statement takes the 'true' branch if the expression evaluates to ANY non-zero value. No trouble need be taken to ensure that it evaluates to minus

one; nor does CB80 compile code to compare it with minus one. In particular, CB80 generates exactly the same code for IF FLAG% as for IF FLAG%<>0.

The CB80 documentation is mistaken on this point: "The IF statement determines whether the expression is true (–1) or false (0) . . ." ([2], p. 51). That description does not even make sense in the context of a two-way branching device. Such a device can work only by splitting the universe into exactly two possibilities: a certain state, and all other states. To put it another way, a conditional ultimately has to branch on a CPU flag; CB80 does what is necessary to make a flag reflect the state of the relation (or the value of the expression.)

CB80 does return either a zero (false) or a minus one (true) when a relational operator appears in an expression, as in DIFF% = (A%<>B%). This is merely to force an integer to mimic the Boolean behavior of a single bit: zero has all bits off, minus one has them all on. Thus one can safely go to extremes like SAME% = NOT DIFF%. It turns out that CB80 generates more code to evaluate such relational expressions than to test relations in IF statements. Still, they have their place, as Mr. Kimbro points out. But it doesn't matter, in his Modification Number 2, what FLAG% is set to, as long as it is non-zero. This being so, Modification Number 3 could have used FLAG% = CHARACTER% – 32 instead of FLAG% = (CHARACTER% = 32), and reversed the test on it; this saves another ten bytes of code.

There was an incomprehensible little paragraph on the subject in an old CBASIC manual that scared me off for a while, I confess. If this is a problem, a few tests will assure you that there is nothing weird about the behavior of integers; they just print funny.

So much for the flag. And so much for conditionals. But I'm not finished with Lowcase$.

Why rebuild the string a byte at a time? That is an efficient way to slow down a program and fragment memory. Why not operate on the formal parameter X2$ in place, with PEEK and POKE? CB80, unlike CBASIC, passes strings to functions 'by value': it actually copies the whole string to X2$ before getting down to business, and you won't step on your original. (The copying occurs inside the CB80 DEFinition, using the pointers pushed onto the stack when the function is called.)

Or do I detect a fear of "negative" integers greater than 32K? There was an incomprehensible little paragraph on the subject in an old CBASIC manual that scared me off for a while, I confess. If this is a problem, a few tests will assure you that there is nothing weird about the behavior of integers; they just print funny. You may have to be careful with conditionals and loops, which look at their signed instead of their unsigned values. But the SADD function returns the genuine address of a string, and when you increment it you really will be pointing at the next character.

My version of Lowcase$ (Listing 1) combines these concepts along with a trivial modification or two. The code is another 20% smaller, but the execution is six times as fast.

Mixed-mode arithmetic

All functions with numeric arguments convert between integer and real data types if you specify the 'wrong' type. If you code COS(I%), CB80 will insert code to convert I% to a real before calling the cosine routine. This should come as no surprise. The same applies to arguments to user-defined functions ([2], p. 34), whether or not the function is EXTERNAL. I mention this because the manual also warns (p. 122) that no type-checking is done between modules (how could it?) — evoking images of such devastation that I forgot about page 34 for a while.

What the manual does not seem to make clear is whether reals are rounded or truncated during conversion to integers. Nor does it specify in what order conversions are made in mixed-mode calculations. Of course, we normally want to avoid mixed-mode, but there are times when it is appropriate. At those times we need to know what's going on; I ran some tests to settle these questions.

To discuss the results concisely, I'll use a special notation:

Icv(r) means convert real expression r to an integer
Rcv(i) means the obvious converse

As an example, the expression R1/(2*I%) is coded as R1/Rcv(2*I%), that is,

```
LD      HL,(I%)
ADD     HL,HL           2*
CALL    ?CRSH           Convert HL to Real
                        on Stack
LD      HL,R1           Pointer
CALL    ?DRMS           Divide Reals, (HL) /
                        Stack
```

or that the statement I% = 2*I% + R1 is implemented as I% = Icv(Rcv(2*I%) + R1), as in fact it is.

The answer to my first question is that Icv(r) does not truncate r but rounds it to the nearest integer, in every case I have tried. The rounding is 'away from zero': 0.5 becomes 1, –0.5 becomes –1, and both 0.4 and –0.4 round to zero. One of two library routines is used:

```
CALL    ?CIHM           Convert (HL) to In-
                        teger in HL
```

when r is a real variable or constant, or

```
CALL    ?CIHS           Convert Stack to In-
                        teger in HL
```

when r is an intermediate calculation.

If you're in doubt as to a particular case, just compile with the %DEBUG I toggle; if one of the above is coded, the real is rounded. They are in fact used for CHR$(real), for array subscripts (if one must use reals for such operations), and, of course, for assignment of a real quantity to an integer variable. I'd be interested to hear of any cases where other library routines are used for Icv(r) — especially if they result in truncation — other than INT%, of course. (I just heard from DRI [5] that all automatic Icd's are supposed to round.) By the way, dividing two integers returns the truncated value automatically; INT%(I%/J%) makes a lot of work to no purpose.

Now for mixed-mode expressions. We already know ([2], p. 12) that any operation combining a real quantity with an integer is real: I% op R1 is coded as Rcv(I%) op R1. CB80 performs the operations in an expression by operator precedence, converting the integer operand of any currently mixed pair to real. To illustrate, the preposterous expression 2*I%+J%+R1+2*K%*L%+M% becomes Rcv(2*I%+J%) + R1 + Rcv(2*K%*L%) + Rcv(M%). Simply putting R1 at the end leads to Rcv(2*I% +J%+ 2*K%*L%+M%) + R1, saving several bytes and much work at run time.

More important than trimming code is the question of getting the result you want. I%/2 is not the same as I%/2.0: for I% = 5, you'll get 2 and 2.5, respectively. And if you assign the second directly to an integer, its value will be 3.

Wish list

The single most valuable enhancement I can think of for CB80 would be automatic coding by the compiler of calculated string constants directly into the static constant area. Assignments like

```
CLS$ = CHR$(1Bh) + CHR$(0Ch)
```

generate a lot of code. The compiler recognizes constants in other situations; couldn't it see that here the programmer just wants the constant string and not the CHR$ function? Another way would be a new notation, like PL/I-80's caret: '↑G' means CHR$(7). Better yet, a hex notation applied to strings, allowing spaces for clarity:

```
PRINTER.RT.ARROW$ = x08 08 08 49 2A 1C 08x
```

Another facility I would like to see would provide for multi-line comments without a REM on each line. Yes, you can extend a REMark over more than one line with the continuation character, but this is usually just as inconvenient. One I like is in Microsoft's M80 assembler: .COMMENT x, where x is the character you specify as the delimiter. All text is ignored until the next occurrence of x.

Bugs in CB80

Here are a few bugs I am aware of in Version 1.4, which are not mentioned in the READ.ME file that comes with Version 2.0. I haven't tested them under 2.0, and I'd be glad to hear from readers of any others, as well as of serious discrepancies in the documentation. I'm waiting to hear DRI's plans as to these bugs, and will pass same along to you.

1. **SADD:** Manual states, p. 98, "If the expression is a null string, SADD returns a zero." But SADD(X$) may return a non-zero value when X$ is null, if it has been read from a file (READ -1; LINE X$). (DRI says documentation should read to the effect that if a string variable has ever been assigned, SADD will be non-zero.)

2. **INPUT:** Manual states, p. 57, "A prompt string must be a string constant." The Programming Guide [3], however, states on p. 40, "The INPUT prompt can be any expression; the first operand must be a string constant." A program will indeed compile without error only if the prompt begins with a string constant; but if the prompt contains any variables the program goes haywire at runtime:

```
INPUT "" + PROMPT$; X$
INPUT "Enter Name if " + NAME$ + " is incorrect:";
  LINE X$
```

(DRI says this bug is known and the documentation will be fixed to prohibit variables in prompts.)

3. **VARPTR:** Manual gives this example, p. 113:

```
DIM A$(10)
CALL PROCESS(VARPTR(A$))
```

But the form VARPTR(A$) will not compile. We are reduced to VARPTR(A$(0)), which is a different, and non-permanent, address. Is DRI deliberately hiding the permanent pointer to an array? Granted CB80's array structure is different from CBASIC's; this seems to me a loss of power. No ambiguity is possible now that CB80 no longer allows the same name for a scalar and an array. Does DRI plan to change the array structure? In any case, I would like to see documentation on array structure. (DRI says VARPTR(A$) will be allowed in CB86—but not in CB80.)

4. **SGN:** Manual states, p. 99, "Real number expressions convert to integers." The reverse is true. Which does DRI intend? (DRI intends the reverse and will fix the documentation.)

5. **CHAIN:** Manual states, p. 21, that we can chain to either an overlay or a directly executable program. Overlays chain fine, but chaining to a .COM file is unpredictable.

```
INPUT "Chain to:"; PROG$
CHAIN PROG$ + ".COM"
```

may produce partial, incorrect, or dangerous operation of the new program. The manual gives no restrictions on the kinds of .COM files to which one can chain. There are obvious limitations on this procedure in the first place (e.g. some programs require command-line arguments) — but it would be a valuable option. (DRI says this will be fixed in Version 2.1.)

A program will indeed compile without error only if the prompt begins with a string constant; but if the prompt contains any variables the program goes haywire at runtime.

6. **User-Defined Functions:**

a. The notorious fact that we get unpredictable results at run-time if we code the same function more than once in the same statement. (One such statement clobbered a disk, although the function called had nothing to do with files.) I considered this one so serious, and knew that at least VanNatta and his readers were aware of it [4] that I expected it to be fixed in Version 2.0. No mention of it in READ.ME, and I'm leery of testing it. (DRI says this bug is not economically repairable and the documentation will be fixed to forbid it.)

b. Argument-passing: First, the Programming Guide, p. 33, is confusing on the subject: all parameters are indeed passed "on the hardware stack" — but it is not made clear how this is done for each data type. In particular, strings are passed by pointer but real numbers are passed by 8-byte value. Second, the fact that the entire eight bytes of a real number get dragged onto the stack leads to a potential bug: 13 real parameters will overflow the 100-byte stack (p. 29), whereas 15 parameters are

allowed (p. 39). Of course, a function with that many arguments of any data type would seem to be a bit pathological. Still, consistency would be nice. If I had my druthers, I'd pass only the pointers to real numbers in the first place — but this is a philosophical point, and no doubt there are other tradeoffs. (DRI says the stack is larger — 200 bytes — in version 2.0.)

References

1. Robert P. VanNatta, *Lifelines/The Software Magazine,* March 1983
2. *CBASIC Compiler Language Reference Manual,* Third Edition, December 1982
3. *CBASIC Compiler (CB80) Language Programming Guide,* First Edition, 1983
4. Robert P. VanNatta, *Lifelines/The Software Magazine,* October 1982
5. Norman Alcott and Paul Lancaster, Digital Research, phone conversation, November 23, 1983.

Table 1

Bytes of code, CPU flag used (Zero, Sign, Carry) for conditionals

	Integer	Real	String
op	expr% op expr%	expr% op k −8<=k<=8	expr op expr expr$ op expr$
=,<>	15, Z	8+k, Z	12, Z
<,>=	13, S	8+k, C	12, C
>,=<	13, S	13, S	12, S

Figures are minimums to load, fix flag, and branch

\Fn Lowcase$; JSC version

```
DEF    LOWCASE$(X2$)    PUBLIC
       INTEGER  LAST.CHARACTER%,  CHARACTER%,
            POINTER%,  FLAG%
       INTEGER  CHR.ADDR%

         LAST.CHARACTER% = LEN(X2$)
         CHR.ADDR% = SADD(X2$) + 2  REM  -> 1st chr of X2$

         FLAG% = 0     REM     No mod 1st chr no matter what
         REM     & start loop with 1 in case there are leading blanks
         REM        ( also guarantees no damage of SADD = 0 )

         FOR POINTER% = 1  TO LAST.CHARACTER%
              REM     Loop is really just a counter now
           CHARACTER% = PEEK(CHR.ADDR%)
           REM  ASC(MID$(. . .))

           IF CHARACTER% >= 41h THEN   \
             IF CHARACTER% <= 5Ah THEN   \
               IF FLAG% THEN   \ Prev was <>" "
                 POKE CHR.ADDR%, CHARACTER% + 20h

           FLAG% = CHARACTER% + −20h  REM  zero iff " "
           CHR.ADDR% = CHR.ADDR% + 1  REM  -> next chr
           NEXT
         LOWCASE$ = X2$
       FEND

REM    Could make this work with punctuation, too:
REM    FLAG% = CHARACTER% − 41h  REM  Neg iff < alpha
REM      also start with: FLAG% = −1,
REM    and change test to:  IF  FLAG% >= 0
REM    No cost in code
```

Editor's Note: All examples and listings are in Zilog mnemonics. In the October 1983 issue (p. 23, "Tips And Techniques," John Coggeshall), the ± sign should be a ↑.

Answer To Last Month's Puzzle

■	1 S	2 T	3 R	U	D	6 L	■	7 H	8 I	9 L	10 L
11 L	U	R	E	R	■	12 N	13 A	P	D	U	E
14 I	T	I	N	E	15 R	■	16 P	O	L	K	A
17 S	U	S	A	N	E	18 S	A	W	Y	E	R
19 P	R	E	L	A	T	U	R	E	■	■	■
20 S	E	C	■	■	■	21 E	T	R	22 E	■	■
■	■	23 T	24 R	25 O	26 D	■	■	■	27 X	28 E	29 V
■	■	■	30 A	R	I	31 S	32 T	33 O	T	L	E
34 D	35 A	36 V	I	S	F	O	U	L	G	E	R
37 A	V	I	S	O	■	38 M	E	D	O	C	S
39 M	E	D	I	N	40 A	■	41 S	E	T	T	E
42 S	C	A	N	■	43 B	44 U	D	N	O	R	■

Feature

PL/I From The Top Down—

by Bruce H. Hunter

In this article we will look at pointers, based storage and elementary data structures. Pointers, based storage and data structures such as lists are generally considered advanced programming techniques and are used for data handling where speed of execution and the dynamic allocation and de-allocation of storage is the prime requisite. We will also be looking at other elementary data structures such as the binary tree — an even more sophisticated programming technique to maintain order in the storage and retrieval of data.

An excellent book on the subject of data structures is Data Structures and PL/I Programming, *by Moshe J. Augenstein and Aaron M. Tenenbaum, published by Prentice-Hall.*

CHAPTER SIX — POINTERS, BASED STORAGE AND LISTS

The concept of pointers is neither new nor unique to PL/I. It is a primary concept in systems level languages like C and PL/M, and is a very powerful tool in programming.

Before you can understand the concept of pointers, you need to understand how data are stored. When a scalar (single value) is stored, it has a unique place in memory, very similar to a post office box. That unique place in memory has an address the same way the post office box has a number. Let's say that we have a variable X with a value of 32. In the following illustration, the unsigned short integer 32 is stored at memory address 24166.

address	contents of address
24166	32

Remembering our variable X, which is assigned the value of 32, it is important to know that programmatically X is associated with the address, not the value. The association between the variable called X and the address 24166 is determined at the linkage process after compilation. In this way the variable X, the address 24166 and the contents of the address 32 are all associated.

Where do pointers fit in? Pointers point to the address of a variable. In fact, a pointer *is* the address. If you ask for the address of X, the address function or operator will return 24166. If you ask for the contents of 24166, you will get 32. Although this is pretty straightforward, the fact that a variable has an address is not necessarily apparent to a novice programmer in BASIC, Pascal or FORTRAN. That knowledge is not necessary in order to program in those languages. The fact that a variable has a name and a value is enough, and the knowledge that an assignment statement like "X = 32" signifies that the value of X is 32 tells you all you really need to know to program most applications in those languages.

You can get your hands on the address in BASIC with the function VARPTR for variable pointers. As I said before, a pointer is simply an address of a variable. However, just because VARPTR exists doesn't mean that it is very efficient. BASIC wasn't intended to manipulate variable pointers with ease, but it can be done. The point I want to stress is that many languages not only make addresses (pointers) easily accessible, the ability to deftly manipulate pointers is built in. In system-oriented languages like C, PL/M and PL/I, pointers even have their own data type.

If you know where a variable "lives," you can do anything you want to it if you can only get your hands on it. Many languages provide you with the ability to get to the address and change the contents. C provides an address operator ('&') to yield the pointer to a variable, and pointer manipulation in C is not only accomplished with ease, it is an integral part of the language. We already discussed BASIC's closely related function, VARPTR, which points to the address of the first byte of a variable. In addition, BASIC's PEEK and POKE allow the contents of an address to be seen and changed. The effect is similar to using pointers. Pascal handles pointers reasonably well by the use of a "↑" symbol. In PL/I pointers are a distinct data type. Like C, PL/I handles pointers with ease. Let's take a look at some PL/I pointer basics.

Pointers in PL/I

Pointers are variables and, like any variable, they must be declared. Keeping with our example of X as a variable, a pointer to X would first have to be declared, like this:

```
dcl X_ptr pointer;
dcl X fixed;
```

At this very early point in the program the "X_ptr" variable is not associated with X, just declared so that storage will be assigned by the program for both variables. To associate the pointer to the variable, the *addr*() (address) function must be used. *Addr*() is a built-in PL/I function that returns the address of a variable. Continuing with our example, the following statement would appear in the body of the code:

```
X_ptr = addr (X);
```

Here the function *addr*() returns the address of X and stores it in the pointer variable "X_ptr." Now you are ready to use the pointer programmatically.

Pointers allow the direct contents of memory to be written, read or altered. They are powerful, and can be dangerous if used ignorantly. A misused or misdirected pointer can blow up a system! In low level languages, pointers are used to directly manipulate data. In high level languages, their use is the same, and additionally they are used to create data structures such as lists and trees. Let's take a cursory look at lists and what their function is.

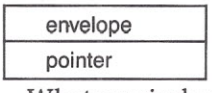
Lists

Data can be organized to make it more accessible by storing the data with one or more pointers to similar data in

such a way as to have one item point to another. Organizing data in this way creates "data structures," and the most basic form of a data structure is a linear "list." The essence of a list is the ability of each element in the list to point to another element. For example, if you have a list of office supplies, you might want to list them in straight alphabetical order. "Adding machine tapes" would lead your list and point to "clips-paper" which would point to "envelopes" which would point to "erasers" which would point to "folders" and so on until you got to "xerox fluid," which would point to nothing. This is a one-way linked list. Now if you added a set of pointers in the other direction and had "xerox fluid" point to "thumb tacks," which would point to "staples" and so on until you had "clips-paper" point to "adding machine tapes" which pointed to nothing (null pointer), you would now have a two-way linked list. What if you kill the null pointer and have "xerox fluid" point to "adding machine tapes"? You now have a circular list.

Nodes

Now let's talk about nodes. A node is at least one piece of data and an associated pointer. Nodes are the building blocks of data structures. They can be simple, having one piece of data and one pointer, or they can be very complex, housing huge masses of data and multiple pointers. In the following illustration, the data element from our list of office supplies is "envelope," and envelope has a pointer to the next office supply item, "erasers." It is a node:



What precisely does this pointer point to? In a programmatic one-way linked list it points to the address of the next data element, "erasers." Repeat, address! Therein lies the key to pointers: they point to the address of something that you have either identified or that is related to something you have already identified.

PL/I's data structures allow the concept of a node to be a reality. Any language that supports both structures and pointers like C and Pascal (type record) can do this. Following is how you would declare a data node for office supplies:

```
dcl
  supply_ptr pointer;

dcl
  1 supply_node based,
    3 supply char (32) var,
    3 supply_ptr;
```

If you know where a variable "lives," you can do anything you want to it if you can only get your hands on it.

The data structure "supply_node" ties the individual office supply item to the pointer to the next office supply item. The character variable "supply" shares the same structure with a pointer to the next supply node. The word *based* refers to a specific storage class, and it tells PL/I that the data, the entire structure, will be stored above the program data area in PL/I's own version of the heap. This is a storage area that has no variable names and no program-allocated storage in the usual sense, just storage that is allocated and de-allocated by the use of PL/I's *allocate*() and *free*() functions.

The essence of a list is the ability of each element in the list to point to another element.

Going back to our one-way linked list, the node with "adding machine tapes" has a pointer that points to the address of "clips-paper" and so on until we get to to the address of "xerox fluid," which has a pointer that points to nothing, the null pointer. The null pointer is an entity that has the useful purpose of telling you that you are out of addresses and therefore at the end of the list. The use of the null pointer is a primary concept in data structures. Many languages guarantee that a pointer to data will never have a value of 0. As a result, this guarantees that a null pointer points to nothing and ensures that it is the end of the list.

Considering our two-way linked list, each node would house an office supply description and two pointers, a right pointer and a left. "Envelopes" would have a right pointer to "erasers" and a left pointer to "clip-paper." "Adding machine tapes" would have a right pointer to "clips-paper" and a left null pointer. "Xerox fluid" would have a left pointer to "thumb tacks" and a null right pointer.

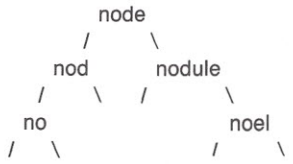
Now that we have pursued elementary data structures like linear lists and some of the basics of data structures like nodes, I want to take a look at another very important type of data structure, the binary tree.

The binary tree

The binary tree is a triangular structure branching from the top down. It starts with a single node, the root node. All nodes have two pointers: a right and a left. The root points to (the addresses of) the two lower nodes which now branch to four nodes, and so forth. This is assuming the tree is full. Going along with the tree analogy, all terminal nodes are called leaves. A node is "father" to the nodes directly below it. Those above are called "ancestors," while those below are called "descendants." The binary tree can easily be the subject of an entire large chapter but we will take a light look to give you an idea of the scope of pointers and what they can do.

Imagine, if you will, a balanced binary tree of all the words in a small dictionary with the root node being what would otherwise be the center of a linked list. Branching all the way down to the left and in the lowest leftmost corner is "aardvark." Branching all the way down to the lowest right corner is "zymurgy." If you were to search the tree starting at the root node, and you were to create a test to see if the node was alphabetically less than (left) or greater than (right), the comparison based on the ASCII collate sequence, you could branch through each level of the tree and only have to read one element. Should this dictionary tree have some 65,000 words in it, it could be searched from top to bottom in 15 tries because it would only have 16 tiers. The

efficiency becomes apparent. If it were a two-way linked list and you started at the exact center, you could have made it with 32,500 tries. You can appreciate the advantage of the binary tree! If by some miracle of coincidence the word "node" were at the center of the list, the root node and its immediate descendants might look something like the example below:



Based storage

Let's review based storage again to understand the based storage class in PL/I. Based storage is the wild and wonderful storage area located above the allocated storage area of the program and extending all the way up through memory to the base of the operating system itself. It is sort of a "no man's land" in that no variable names are used there in the usual sense. Data stored there can be reached only by pointers, not by the use of variable names. Each data structure stored there is said to "have a template fit over it." The "template" is established by the declaration of the node and does not allocate storage but rather defines it. Since each node (or "variable') can be reached only by direct addressing, the pointer to the address is called the base — hence "based storage."

The following data types can all be based storage:

```
Char()
Pointer
Fixed
Bit()
Float
Dec
```

Additionally, they can, and usually will, have a pointer reference. The pointer reference is the cement that holds the whole based system together. Without a pointer to a data item, that piece of data is inaccessible. The pointer reference is the pointer to the data and that makes the mechanism of based storage run. The following declaration declares a character variable "foo" with a pointer reference "p." The declaration ties the variable and the pointer reference together:

```
dcl
  foo char (32) based (p);
```

This declaration makes "foo" a pointer character variable, and "p" is the implicit base or pointer. Not that based storage cannot be created by declaration without a pointer reference. The following declaration has no pointer reference:

```
dcl
  bar fixed based;
```

Here the variable "bar" has no pointer qualifier.

"Foo" has a pointer "p" associated with it, but "bar" must be referenced every time it is used. We must get a pointer to reference "bar." The mechanism for this is an interesting looking device formed by the minus character followed immediately by a right angle bracket ("->"). It is called an arrow and is used "as a separator in a pointer qualified reference," to quote the manual. For example:

```
ptr -> bar
```

The pointer qualifier "ptr" now points to "bar."

We now have declared a based variable and even set a pointer to it, but we're not done yet. It is necessary to allocate storage space for it. A static variable will have storage pre-allocated and an automatic variable will have space automatically allocated by the system all because they exist in the program's data area. However, based storage must have its space allocated programmatically. This is because it is based (heap) storage. It gives nothing and takes away nothing unless it is explicitly asked to. It is not an easy concept to deal with, but the first thing you need to know is that you do not get space in the heap or based area unless you specifically and deliberately ask for it. To allocate storage you do this:

```
allocate foo set (p1);
```

Allocate sets aside sufficient storage for "foo" and sets the pointer 'p1" to the address allocated to "foo." The *set* attribute of the allocate statement sets the pointer to the data. This space will remain in existence for the life of the program or until it is freed. The statement:

```
free p1 -> foo;
```

frees the space occupied by data called "foo," "p1" having shown the system the address.

Pointers can be set to an address in the normal program storage area (data storage or non-based) as well. The function ADDR() returns a pointer value, the address of a variable.

```
p = addr (foo);
```

This returns the address of "foo" and stores it in the pointer variable "p."

There are times when a pointer must be set to an absolute address such as a data port for a modem or some other device. Pointers are machine words, unsigned integers, and as such are not a regular storage class in PL/I as they are in C. Fortunately, the *unspec*() function is always available to convert bit strings into usable machine words. In the example below, the pointer "p" is set to the warm boot address:

```
unspec (p) = '0001'b4;
```

The unspec function, here used as a pseudo-variable on the left of the assignment, converts the bit string on the right into a true binary value.

Putting pointers to work

As you can see, the potential power of pointers is virtually unlimited. With pointers, you can go anywhere in memory, read anything, and put anything into memory. They are as dangerous as they are powerful. Additional food for thought: if gotos can obscure a program, imagine what can be done with poorly defined pointers! Enough philosophy. Let's look at a program using based storage, nodes, lists, and pointers.

LIST is a program to read a file of accounts, store the account name and number in a linked list, and then display the list to the screen. Although it is a freestanding program, it could be a segment of a larger program such a chart of accounts program, and it could be used to store the names and numbers for display at a later time. The purpose of storing the name and numbers in based storage is to keep from having to pre-allocate a massive block of storage for an array since based storage can be allocated dynamically.

►

```
                    /* LIST */
          /* A program to store account names and numbers */
/

list:
   proc options (main);
   dcl
      (p, p__start, p__now, p__last) ptr,
      /*
      */
      1 act__node based (p),
         3 act__name char (32) var,
         3 act__nbr fixed,
         3 p__next ptr,
   % replace
      CLEAR by '↑L',
      TRUE by '1'b;
   dcl
      accounts file,
      i fixed,
      1 act__structure,
         3 name char (32) var,
         3 number fixed,
         3 address char (64) var;

   put list (CLEAR);
   put skip list ('↑l↑ldisplay accounts');
   open file (accounts) direct input env (f (128));
   call read;
   call display;
   /*
   */
   read:
      proc();

      on endfile(accounts)
         goto endloop;
      allocate act__node set (p);
      p__start = p;
      read (accounts) into (act__structure) key (1);
      act__name = name;
      act__nbr = number;
      p__last = p;
      do i = 2 to 32767;
         allocate act__node set (p);
         read file (accounts) into (act__structure) key (i);
         act__name = name;
         act__number = number;
         p__last -> pstore = p__now;
         p__last = p__now;
         end; /*do*/
      end__loop:
      p__now -> p__store = null;
      end read;

   display:
      proc();
      dcl
         p__last ptr,
      i fixed;
      put list (CLEAR);
      put skip (3) edit ('↑l↑l Accounts') (a, 2(skip));
      p = p__start;
      do while (p__store ↑= null);
         put skip (2) edit (act__number, act__name)
            (f(6,0), col(12), a);
         if mod (i,10) = 0 then
            do;
               put skip list ('enter to continue');
               get skip(2);
               end;
         p__last = p;
         p = p__store;
         free p__last -> act__node;
         end; /*dowhile*/
      end display;
   end list;
```

Now we'll examine the program a section at a time, starting with the declarations:

```
list:
   proc options (main);
   dcl
      (p, p__start, p__now, p__last) ptr,
```

"p," "p__start," "p__now" and "p__last" have been declared pointers. The pointer "p" will be associated with the based data structure "act__node."

Here's the declaration of a node:

```
   1 act__node based (p),
      3 act__name char (32) var,
      3 act__nbr fixed,
      3 p__next ptr,
```

The structure "act__node" now with a base "p" contains the account name and number and a pointer "p__next" that will point to the next node. This data structure is the node.

Next we get into some routine housekeeping and an ordinary data structure declaration:

```
   % replace
      CLEAR by '↑L',
      TRUE by '1'b;
   dcl
      accounts file,
      i fixed,
      1 act__structure,
         3 name char (32) var,
         3 number fixed,
         3 address char (64) var;
```

A file CALLED "accounts" now exists consisting of records with the fields "name," "number" and "address" as a data structure.

Next the file "accounts" is opened for input (read only). There are two procedures, "read" and "display," which are called by the main:

```
   put list (CLEAR);
   put skip list ('↑l↑ldisplay accounts');
   open file (accounts) direct input env (f (128));
   call read;
   call display;
```

In the read procedure, the endfile condition will cause program execution to go to the bottom of the loop:

```
   read:
      proc();

      on endfile(accounts)
         goto endloop;
```

Then based storage is set aside for the account node with "allocate," and the pointer "p" is set to it.

```
      allocate act__node set (p);
      p__start = p;
```

Remember "p" was associated with the node in the initial declaration. Since this is the first pointer to have an address assigned to it, the pointer "p__start" is set to this address so the beginning of the list can be found later.

The account name and account number are extracted

from the file record and assigned to the first node:

```
read (accounts) into (act_structure) key (1);
act_name = name;
act_nbr = number;
p_last = p;
```

The end pointer "p_last" is assigned with the location of the node just in case it is the last node (unlikely but not impossible).

Then a do-forever is initiated, and at each iteration a new node is allocated and its address stored in the pointer "p_now."

```
do i = 2 to 32767;
  allocate act_node set (p – now);
  read file (accounts) into (act_structure) key (i);
  act_name = name;
  act_number = number;
```

Now we have come to the tricky part. The pointer within the node "p_next" is intended to store the address of the next node:

```
p_last –> p_next = p_now;
p_last = p_now;
```

The last node written must have its internal pointer "p_next" set to the address of the newest node. So the notation

```
p – last –> p_next = p_now;
```

says to take the pointer "p_next" pointed to by "p_last" and assign it with the value "p_now." The pointer "p_last" which holds the last used address is set to "p_now."

The last record has its stored pointer address "p_next" set to null, so when the list is searched, it will flag the end of the list:

```
  end; /*do*/
end_loop:
p_now –> p_next = null;
```

The pointer p associated with the node is given the address of the first node or starting node.

```
  end read;
display:
  proc();
  dcl
    p_last ptr,
    i fixed static init (1);
  put list (CLEAR);
  put skip (3) edit ('↑I↑I Accounts') (a, 2(skip));
  p = p_start;
```

The loop, a do-while, will now run until it sees the stored pointer within the node "p_next" go to null:

```
do while (p_next ↑= null);
  put skip (2) edit (act_number, act_name)
    (f(6,0), col(12), a);
```

This puts the account name and account number in columns 0 and 12, respectively.

Now we have the old scroll-stopping trick:

```
  i = i + 1;
  if mod (i,10) = 0 then
    do;
      put skip list ('enter to continue');
      get skip(2);
      end;
```

What this says is if the number of iterations, "i," is an even multiple of 10, stop the program execution until a carriage return (enter) is received, or in this case, two carriage returns.

Before going to the next iteration of the loop, we set the pointer "p_last" to the address of the present pointer associated with the node being presently read, "p":

```
p_last = p;
p = p_next;
free p_last –> act_node;
```

Now the pointer associated with the node is set to the location (address) of the next node, p_next. The last instruction does the housekeeping. The last node read pointed to by "p_last," is freed or released from storage.

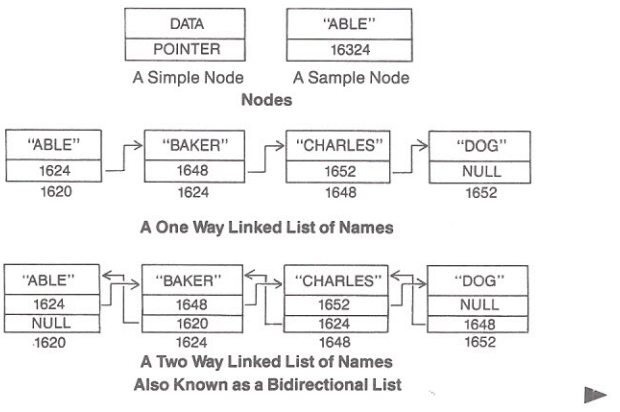
The last three program lines simply wind everything up:

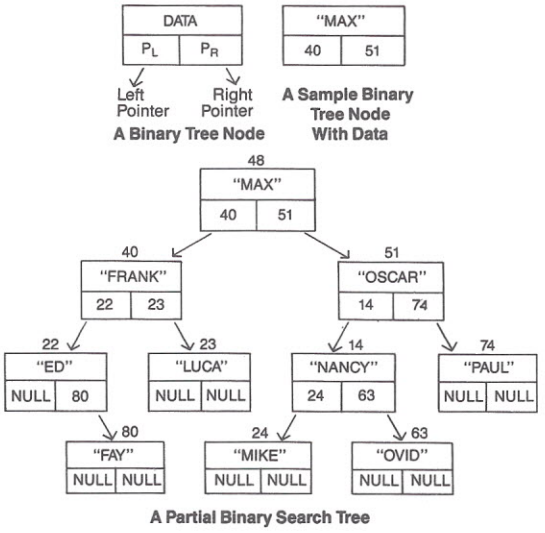
```
    end; /*dowhile*/
  end display;
end list;
```

The LIST program deals with a one-way linked list with the pointers aiming forward into the data. It would have been much easier to program and more understandable to read if the pointers pointed back to the first node, but what would have the effects have been? The list would have read from the last entry to the first, hardly what is required from the majority of lists. If you were to write a "do all" list module, it could be well worth the effort to set a pair of pointers, one back and one forward, for a two-way linked list. Then the data could be read in either direction, and this would make it applicable in more programming situations.

Data structures are an advanced concept that allows the programmer to reach another plateau in programming. A somewhat related concept in data handling as it applies to file records is keyed files. Just as the pointer points to a piece of data in memory, a key points to a record in a file. I will deal with keys and keyed files in the next installment.

With pointers, you can go anywhere in memory, read anything, and put anything into memory. They are as dangerous as they are powerful.





A Partial Binary Search Tree

SOFTWARE NOTES

C-Systems C-Window

by Ron Watson

Release 1.2 of c-systems c-window, a source level debugger, has some new features that will warm the cockles of a hacker's heart.

The previous version was quite impressive, allowing debugging commands to be entered in C language syntax, and referencing the source program variable names directly. This removes the need for the machine language listings necessary to locate program parts by their hardware addresses.

Version 1.2 has added an even fancier wrinkle. When a program operating under control of this version is running, the screen is divided into three horizontal windows. The topmost window is for commands to and messages from the debugger; the center window is for the application program input and output; and the bottom window displays the source program, showing five or six lines around the line currently being executed. This, of course, requires that the source file be on-line and accessible to the debugger, but the program will work if the source is not available; it just eliminates the bottom part of the display.

C-window provides a generous set of commands that allow for variable display, variable alteration, single step and trace, and break points.

Variables are displayed or altered using their source program names and may be accessed in decimal, hexadecimal, character and string formats. Nearly any valid C language expression may be entered, which is then evaluated by the debugger and displayed in the upper window. A variable is altered by entering an assignment statement.

Besides the single step, trace and break-point commands, it is also possible to establish an "expression break point." You do this by entering a C language expression which is then evaluated prior to the execution of each statement. When the expression evaluates as true, a break occurs. This can be particularly useful when trying to locate errors in subscripts and pointers, as it does away with the need to set break points in the lower levels of nested loops and manually display the program status. Instead, the expression break can be established to watch for a certain condition and stop the program when it occurs.

Another productivity enhancement in c-window is a feature called automatic commands. These commands are entered into the debugger and executed automatically each time a single step or break point is encountered. They can be very handy when it is necessary to display several variables each time execution is interrupted, removing the need to retype the display commands each time. Any c-window command can be entered as an automatic command, and up to five automatic commands can be active at any time.

The overall command set is very complete, allowing for the establishment, clearing and display of each command, and it is quite easy to find out what you've told the debugger to do, and change or re-establish any parameter.

C-window will add about 30K to the execution-time size of the program being tested. On smaller machines it may be necessary to set up stubs or shells to debug larger functions, but this is good programming practice in any case.

By way of criticism, it is possible to crash the debugger by entering improper commands, but they must be very subtle in their impropriety. If illogical activities seem to be occurring, you just end the debugging session and start over, taking extra care when entering commands that alter the contents of variable storage areas.

C-window only works with programs compiled with the c-systems compiler, and the version that displays source code is currently available only for the IBM-PC. The compiler and debugger are available separately for $195 each, and there is a c-window demo package you can buy for $45. Warning: don't buy the demo unless you are prepared to buy the whole works; the demo is really impressive.

Review

TK!Solver

by Ron Watson

Every day brings an avalanche of mail announcing "new" programs for the IBM-PC. Most are remakes, supposedly improved, of existing applications; the fingers of one hand are an ample computational device for counting the real innovations that come along in any six-month period. TK!Solver represents one of those rare occurrences.

This program is as original in concept as Visicalc was when it was first introduced. I showed the program to about a dozen people, explaining briefly what it could do, and got an enthusiastic response from everyone until I asked what, specifically, they would do with it. Each agreed that it was a great tool, but had no idea how it might be used to solve a problem that wasn't already being solved in an adequate fashion. This indicates, I believe, a certain lack of imagination on their part, due in large part to the generality of TK!Solver. I suspect Visicalc initially met with the same reaction in many quarters, until the more imaginative (or more desperate) users had an opportunity to explore the program's possibilities.

In a nutshell, TK!Solver is a general purpose modeling system. So, you might respond, is any good spreadsheet program. True, but spreadsheets require you to specify your model in arbitrary terms, and can be as difficult to decipher six months later as any custom-written program. They are also very clumsy when you need to compute the same function with several, or several hundred sets of variables. Nor do they have good facilities for conversion of units of measure, and no method of resolving functions with more than one solution. TK!Solver addresses all of these requirements.

The program works by providing several "sheets" or screens, which are meant to be analogous to the sheets of paper one might use to solve the problem manually. Each sheet has a defined purpose: there is a "rule" sheet on which to specify the equations to be solved, a variable sheet which lists the variables and their values, a "units" sheet which can be used to specify conversions between various measurements, etc.

You begin by specifying the relations or equations to be solved on the rule sheet. This is done in the usual algebraic notation. To calculate the gas mileage for a trip you might enter an equation like:

mlg = d/gas

meaning that the mileage (mlg) is found by dividing the distance traveled (d) by the amount of gas used (gas). As this function is entered on the rule sheet, the program enters the variable names (mlg, d, gas) on the variable sheet. You could then move the cursor to the variable sheet, enter values for distance and gas, hit the ! key and see the calculated mileage appear alongside the mlg variable. You could also, with only a little additional effort, enter a list of values for distance, another list for gas and request that the equation be solved for each pair of values, the consecutive results being stored in yet another list. Then, with just a few more keystrokes, you could see a line plot drawn from the data in the three lists.

To expand on the problem, you could add another equation to the rule sheet, specifying the cost of the trip thus:

cost = gas*price.

Each of these new variable names would be added to the variable sheet, and you could enter a value for price, strike the ! key and see what the trip would cost. Going back, using the same equations, you could enter values for cost, price and distance, recalculate, and see what kind of gas mileage would be required to make the trip on a certain budget. Lastly, if you enter the proper exchange rates on the unit sheet, you can display the cost in pesos or Canadian dollars, the mileage in kilometers/liter and the price in Kruggerands.

Now this example is admittedly trivial, but it should begin to convey the idea of how TK!Solver works. Besides problems that can be solved by direct substitution of values, the program also incorporates an iteration routine that will solve simultaneous equations when there are fewer equations than unknowns.

The commands used to put the program through its paces are reminiscent of Visicalc, and easy enough to learn, even for a novice computer user. The program works in what I would call an "intuitive" fashion: when in doubt, try the most logical thing. It will probably work or lead to a message that will help. The documentation is superb, with an excellent tutorial section that is accompanied by samples on the distribution diskette.

Software Arts is also marketing templates that can be loaded from disk in much the same fashion as spread sheet templates, and put to immediate use in specific areas. I have seen the ones designed for financial management, mechanical engineering, and introductory science, and they are all thorough and easy to use. The introductory science templates should be on every science student's Christmas list; it could make physics and chemistry homework a much pleasanter experience. This, by the way, could be the most valuable application for this program. It is not sold as an educational tool, but imagine the aid a science or engineering student could get in understanding a principle such as the relation between distance, rate and time in the d = r*t formula if he could use this program to generate a graph showing the relationship. The ease with which experimental data can be applied to the formula could encourage students to look more closely at their studies.

The list price for TK!Solver is $299, and you can get it for under $200 from your friendly mail-order discount supplier. Buy it and let your imagination experiment with the possibilities.

YOU SPENT $4,000 ON A PERSONAL COMPUTER. FOR ANOTHER $12.50, YOU CAN GET YOUR MONEY'S WORTH.

Today's personal computers have an extraordinary range of capabilities.

For a variety of reasons, however, many business people

are unaware of just how much their computers are capable of.

As a result, they aren't realizing the full potential of their investment.

THE KEY TO GREATER PRODUCTIVITY IN A WORD: SOFTWARE.

Computers do the work. Software does the thinking.

Expanding the amount of work a personal computer can do is merely a matter, then, of gaining access to a broader array of software.

And the software programs available to business and professional people number in the *thousands.*

But where do you go to find them?

THE KEY TO SOFTWARE IN A WORD: *LIST.*

LIST is the first publication that puts software first.

It contains articles by some of the most respected names in the computer field. Written to help you get the most out of your personal computer. No matter what brand it is. No matter what you need it to do.

More importantly, *LIST* contains the *LIST Software Locator,*™ a comprehensive guide to over 3,000 personal computer programs—conveniently indexed by application, industry, operating system and hardware. You'll find detailed descriptions of applications software that pertains specifically to the type of business you're in. And the type of needs you have.

LIST is sold at leading computer stores and bookstores. Or, you can phone our toll-free number (1-800-821-7700, Ext. 1110) or send in the coupon below, and receive a copy by mail. The price, exclusive of postage and handling, is $12.50.

Which, when you think about it, is a pretty small price to pay for something that can maximize a much larger investment.

LIST *is published by Redgate Publishing Company, an affiliate of E.F. Hutton.*

I'D LIKE TO GET THE MOST OUT OF MY PERSONAL COMPUTER.

Please send me _____ copies of *LIST* at $12.50 a copy plus $2.00 each for postage and handling. (Tax will be added where applicable.)

☐ VISA ☐ MasterCard (Interbank No. _____)

Card No. _____ Exp. Date ____

Signature _____

Print Name _____

Address _____

City _____ State _____ Zip _____

Send to *LIST,* Redgate Publishing Co., 3407 Ocean Drive, Vero Beach, FL 32960.

Or phone, toll-free: **1 800 821-7700 Ext. 1110**

MasterCard VISA

LIST™

The Software Resource Book For Personal Computer Users

Pipeline

by Jon Wettingfeld

Datapro Announces Comprehensive Micro User's Survey

If you're considing buying a new micro system, or new software for your existing machine, you may find Datapro Research Corporation's new comprehensive survey of hardware/software users extremely valuable. Datapro, a division of McGraw-Hill, specializes in providing information about computer-related areas, as well as the entire business equipment industry. "Datapro Reports on Microcomputers," their most recent report, presents data collected from the rank and file microcomputer end users. It surveys how they feel about the products and services provided, and analyzes the data.

The user segment of the report covers 126 models from 57 manufacturers, 19 primary operating systems, 13 of the most popular software packages, and 90 of the most popular peripherals (modems, printers, plug-in expansion cards, etc.) The results are very interesting and useful, though not always unexpected.

Among personal computers, the two most popular systems used by survey respondents were the Apple II plus (19.2% of the total responses) and the IBM-PC (13.7%). Eight other well-known systems were also ranked just below.

In terms of overall satisfaction, the majority of users responding to the survey said they were highly satisfied with their equipment. 78% said they would recommend it to others. Users also gave their systems high grades for overall satisfaction, ease of operation, keyboard usability, hardware reliability, cost/performance ratio and ease of expansion. Lowest marks (not suprisingly) were given for quality of documentation and speed of disk access.

Among the six types of software categories surveyed (operating systems, wordprocessing packages, electronic spreadsheets, database/file management software, accounting and communications packages), only the accounting software received a "fair" rating. All the others were viewed as "good." Significantly, the single largest factor decreasing overall satisfaction was vendor support. Of the operating systems, Apple DOS received the most responses (26.3%), followed by CP/M (23%) and PC/MS DOS (19%). The most popular wordprocessing package was Micro Pro's Wordstar with 45% of the responses. Apple's Applewriter followed with 20%, in part attributable to that vendor's lead in responses for system units.

The electronic spreadsheet market turned out to be dominated by Visicorp, Sorcim and Microsoft, which jointly accounted for 80% of the total. Interestingly, the Lotus Development Corporation's Lotus 1-2-3 program captured 7.2% of all responses, though only commercially available for six months at the time of the survey. This, according to Datapro, may indicate a preference for spreadsheets with built-in graphics.

In the database/file management category, the lead was held by Ashton-Tate's dBase II, with 40% of the total responses. Among accounting packages, the leading vendor was Radio Shack.

User responses to peripherals were broken down into four product categories:

1. **Multi-Purpose Plug-In Cards:** The market was dominated by Apple compatible vendors: Microsoft and Mountain Computer. Leaders for the IBM-PC were A.S.T. Research, which manufactures expansion cards exclusively for the IBM, and Quadram, which supports both the IBM and Apple.

1. **Display Monitors:** Most customers now purchase monitors from their system vendor. As such, Apple received the most responses, followed closely by IBM. Strong showings were also made by Zenith and Amdek.

3. **Printers:** Epson America received better than 40% of the responses.

4. **Modems:** The most frequently cited among the modems was the Hayes Microcomputer Products modem (54%). Of the respondents whose system had communications capability, 32% said they had participated in a Local Area Network. 59% of those with communications capability used 300 baud modems, and 38% used the 1200 baud type.

Among the six types of software categories surveyed only the accounting software received a "fair" rating. All the others were viewed as "good."

Detailed results of the survey will be made available in printed form to subscribers of "Datapro Reports on Micro Computers," the company's newest looseleaf information service. Subscribers receive monthly updates which are issued in a uniform format to facilitate comparisons. Interestingly, survey results are available directly to computer users through the services of Datapro Online, a computer database retrieval system from Data Resources Inc., also a McGraw-Hill company. Users will be able to search and manipulate data to generate information required in text and graphic formats.

Additonal information on Datapro Reports is available at 800-257-9406. Information about Datapro Online can be obtained through Data Resources Inc., at 617-861-0165.

►

Small Bytes

Digital Research recently announced the availability of its **Graphics System Extension (GSX)** software for PC-DOS and MS-DOS. Already available for CP/M 80, 86 and concurrent CPM, GSX extends an operating system to include graphic output and input functions. According to Digital Research, GSX applications programs can be ported from one computer to another, much the way CP/M provides portability for other applications. Further, because GSX produces interfaces to graphics peripherals, it allows the operating system to control a wide range of graphics devices. The company is said to be hopeful that with consistent graphics interface across all operating systems, the GSX will move towards becoming an industry standard.

Teleram Communications Corp., White Plains, New York, has just completed an agreement for distribution of its **T3000 Portable Computer** with Bubble Memory. Distribution will be by Hamilton Micro Systems, an Avenet Company which is a major re-seller of Micros. David Peterschmidt, Hamilton's director of sales and marketing, has observed that portable computers are the fastest growing segment of the personal computer market, and notes that the T3000 is the "ideal computer for field and office use."

The machine is light in weight and competitively priced. A major innovation is its bubble memory, which enables it to operate dependably in varying environmental conditions. The T3000 is also CP/M compatible, giving the user access to the wide array of software applications packages already on the market for this system.

Review

Univair
Legal Time And Billing

by Robert P. Van Natta

One of the great challenges in the microcomputer world has been the effort of microcomputer programmers to produce a billing package that would meet the needs of lawyers, and the like. The job is doubly difficult for a number of reasons. Some of those reasons are as follows:

1) The task of accounting within a law office is much more complex than anyone, including the lawyers, believes.

2) The amount of data to be managed will vary widely depending on the habits of the particular attorney, and the type of business one has.

3) Few lawyers understand microcomputers and few microcomputers understand lawyers.

4) A basic personality characteristic of an attorney includes the tendency to insist on "doing it my way."

5) Typically, even attorneys within a single law firm cannot agree on how books should "properly" be kept.

6) Ethical obligations within the Bar Association often impose restrictions which greatly increase the complexity of the accounting requirements.

In order to come up with an accounting system that will be satisfactory in a law office one only need design a system that will satisfy two elderly curmudgeons (the senior partners) who both agree that a manual adding machine is plenty of accounting equipment, but who have for the last past twenty years been unable to agree as to whose name should appear first on the stationery.

Curmudgeon number two will want to know if the system is capable of printing his name first on odd-numbered bills (in case he might want to try that someday), and curmudgeon number one will be far more concerned with whether the terminal will highlight his name whenever it appears on the screen.

With this background, you perhaps begin to understand the task that besets any mortal programmer who undertakes to produce a lawyer's accounting system. The Univair package (which is marketed by Lifeboat) is one system that attempts to rise to the task.

Just to set the record straight, you should know that I am a licensed curmudgeon, so if it seems that I am taking a perfectly good software package and picking it to pieces you are entitled to assume the problem is just my curmudgeon-ness showing.

The particular product under review here is the Univair legal accounting system version 2.01, which is written in CBASIC2 and which lists in the Lifeboat catalog for $950. With the appropriate runtime CBASIC package, it is suppose to work with CP/M-86 as well as CP/M-80. The catalog says you need 48K of user memory and at least 482K of disk storage. These figures somehow seem optimistic to me, but if that means two 482K disk drives, I would not flinch quite so much.

The actual test bed that I used was a Radio Shack Model 12 which was souped up with 128K of memory and a bank-switched version of CP/M 2.2 sold by Aton International, 1765 Scott Boulevard, Suite 119, Santa Clara, CA 95050. The disk storage on this machine was the standard 2.5 megabytes characteristic of this machine (two each Tandon 848-2 floppy drives).

The product

The system was delivered to me on a single 8-inch disk together with a manual reminiscent of those that came with early versions of WordStar. The white, three-ring binder is unlabeled on the outside and lacks an index on the inside. It does have a table of contents. Notwithstanding the snide remarks, however, the manual is reasonably well-organized and contains a screen-dump of most, if not all, of the numerous menus generated by the program. In particular, the manual provides helpful information of a tutorial nature in

working through the system and, at the same time, provides enough information about the program to provide a prospective user a chance to guess whether the product has a chance to meet his particular needs.

When it comes to installing this system, Univair did a more-than-adequate job.

Installation and hardware

When it comes to installing this system, Univair did a more-than-adequate job. The terminal menu includes 40 common terminals plus a "user-defined terminal" for the oddballs. The system only uses the most rudimentary terminal control features so it should work equally well on nearly any 24×80 terminal. Reports are produced on 80-column paper although the manual specifies a 132-column printer is required.

When it comes to drive mapping and disk storage, the job is well done. On a floppy installation, up to four drives may be mapped in; however, only two are required. On the two drive configuration, an occasional disk change is required. When worse comes to worst, multiple data disks are also allowed. No capacity statistics are provided, but a quick examination of the data files discloses that the major data files are: 1) customer list with 200 bytes per customer; 2) a case header file with 180 bytes per case; and 3) a charge file with 55 bytes per charge.

I translate this to mean that a couple of megabytes of storage will go quite a ways (but probably not as far as everybody thinks).

Organization and design

The Univair Legal accounting package impresses me as having been conceived by someone who had at least some understanding of the kinds of accounting problems that lawyers actually have. The system works by managing a series of random access data files. The hierarchy of files include a master attorney file, a client list file, a case header file, a charge file, and a payment file.

Before a charge can be entered, the attorney, the client, and the case must be identified. Payments are posted against cases, but are not matched against particular charges within the case. All files appear to be accessed by a binary search routine. Index files are generated for the client list both numerically and alphabetically in order to avoid reorganization of the master client list. The other files are regularly reorganized to maintain them in order for the binary search routine.

All charges are identified by a charge code, which in turn translates into some narrative description at billing time. There is no way to directly make a descriptive comment.

Use and usefulness

Overall, this package does most of what a time and billing package ought to do. It will produce an adequate number of reports and, on command, will generate the bills. The system handles flat rate billings and hourly billings with equal ease.

The system is designed with excellent flexibility. For example, each attorney may have any number of clients. Each client may have any number of cases, and each case may have any number of charges and any number of payments.

For those who want the system to do things other than strictly accounting, there is provision for the entry of CASE MEMOS and for ATTORNEY SCHEDULING. Provisions are also made for connection to the Univair General Ledger. The only thing that is missing is a good system to keep track of the client trust account. Codes of ethics in many states require attorneys to maintain funds that are collected for attorney's fees, but not yet earned, to be kept in a client's trust account, which of necessity must be a separate bank account from the lawyer's general account. From a bookkeeping standpoint, credit balances must be kept in one bank account and must be transferred to another when earned. Ideally, a lawyer's billing package would contain facilities to maintain and reconcile this client's trust account, together with a procedure to search it for funds each time a charge is entered. A cumulative amount of the transfers then needs to be generated so that at the end of a posting session a check may be written to physically accomplish the transfer from the trust account to the general account. The Univair system has some primitive facilities for handling credit balances, but they fall substantially short of meeting the need that I just described. I mention this because the frustration level of lawyer and client alike can rise very quickly when a bill goes out to the client when the funds to cover it are languishing in the trust account.

Built-in features include (almost) automatic index file recovery and facilities for dividing up the data files should it come to pass that they outgrow your data disk. To me, the presence of this "back door" is a highly desirable feature. I have seen all too many accounting systems that lack this "back door." The result is that the hapless user gets his computer and his program going just fine and then one day finds that his disks are full. He must then delete his data and start over again or hire a computer guru to develop a custom file splitting and downloading utility.

Fortunately, this disgusting characteristic of many accounting systems is cleverly overcome with a menu-driven file splitting routine. This goes a long ways towards keeping the system usable on a floppy-based computer.

The system is designed with excellent flexibility.

Dislikes

In working with the Univair Legal Time accounting package I found two trivial things that I didn't like and one serious problem. First the trivia:

The system has, of necessity, numerous menus. Some require that you select the number and then hit the return key. Others respond immediately to the numeric selection. I found this inconsistency unnerving.

The other trivial problem which is potentially more serious is that the input loops lack an escape routine. As is typical with a menu-driven accounting system, the routines to enter a record, whether it is a new client

or a new charge, involve the entering of several fields of data. On completion, the user is asked if he wants to change any of the fields. My gripe is that you have only the choice of changing a field or saving the transaction. There should be a third option of abandoning the transaction altogether. I have been supervising accounting activities on microcomputers for several years, and it is my observation that even the most experienced operators occasionally start to enter a transaction which they realize should not be entered at all. With this system, once you start an input loop, there is no way out until you finally enter a transaction of some sort. It is my opinion, and, possibly, I stand alone in this regard, that no accounting program should have any "traps" that won't allow the user to "ABANDON AND EXIT" at any time. Enough errors get into an accounting system without forcing a user to record something that he already knows is an error before he records it.

My third criticism, and most serious to my point of view, is just plain old speed (or lack of it). I will tell you what I did and you can decide whether my test was a fair one or not, and whether you agree with my feeling that the system is too slow. As you might have guessed, I am a lawyer.

Built-in features include automatic index file recovery and facilities for dividing up the data files should it come to pass that they outgrow your data disk.

For what it is worth, I am a member of a two-person lawfirm. My own accounting system (which isn't Univair) happens to have 2500 names in the client list at the present time. My benchmark was to simply download (using a simple utility that I wrote myself) the 2500 name file that I happened to have into the Univair system. I then brought up the Univair program and waited 38 minutes while it automatically regenerated index files to work with my newly-entered client list. The reason for the great delay was accounted for by the need to sort the file twice. In order to function, Univair requires that there be an index file in alphabetical order, and also one in client number order. Even with the 62K TPA operating system that I used, three passes were required to accomplish the sorting.

Once I got the system going, I hoped my problems would be over. Far from it! I found that adding names to the client list required delays between entries which varied from a few seconds to a minute and a half, depending on the number of names added at one time. I also found that exits from various menus often required delays for sort routines and that an attempt to exit the program always seemed to require a rebuilding of some of the index routines which seemed to take about five minutes. Similarly on start up, the file verification (which was optional) required about five minutes simply to determine if the files were present and in order. I found that the end of month routine required 55 minutes to execute.

Needless to say, I am disappointed. I happen to subscribe to a design philosophy that every computer program ought to be so designed that the operator never (or at least almost never) has to wait over two seconds for something to happen. When the delays occasioned by sorting and crunching are counted in minutes, instead of seconds, I start to get excited (or is it impatient?).

The tragedy of the matter is that I am fully convinced that there is no good reason why this poor performance ought to exist in this system. There are exactly two reasons (I think) why this system is slow.

The main reason revolves around the data management routines. The file management routines evident in this system are of a traditional, but crude, binary type. A binary search is very efficient, and will locate a given data record very quickly.

I have no problem with that. The drawback of such a routine is that for it to work, either the data file, or at least an index file which points to the data file, must be repeatedly sorted.

Programmers often avoid the burden of resorting the file after each data item by making provision for a temporary surge file somewhere. The sort is then triggered on the exit from the routine. This accounts for the great delays in reloading some menus in this and other similarly designed systems.

If it wasn't for the fact that there is a better way, I wouldn't squawk about this traditional method of file management. The better way, of course, is to use B-Tree file management routines. There are a number of commercial assembly language subroutines available that handle B-Trees very nicely and which would be naturals for this billing package. One is FABS from Computer Control Systems, 298 21st Terrace S.E., Largo, FL 33541, and the other is ACCESS MANAGER (AM-80) from Digital Research. FABS has the advantage of being a little easier for the programmer to use, and is compatible with both CBASIC and CB-80. It has the disadvantage of requiring royalties to be paid for its use. AM-80 has an AM-86 counterpart and works with CB-80 and CB-86 only and not with the older P-code compilers. (In fairness I should also say that AM-86 has only been recently released, and AM-80 has only been available since the fall of 1982.)

The advantage, from a user's standpoint, is that such routines make it possible to eliminate most, if not all, of the sorting and file reorganization routines. This is true because the nature of a B-Tree index scheme revolves around the ability to 'INSERT' new keys into the index file in their proper place without reorganizing the whole file. B-Trees are, of course, a bit mystifying, but from a user's standpoint they function just like a familiar Rolodex or card index. New entries may be added or old entries deleted very quickly (usually within my two second standard) and searches for existing entries can be executed with astounding speed.

The second reason the program I got performed much more slowly than it ought is because it is compiled in the now somewhat dated CBASIC P-code compiler. The manual refers to the existence of CB-80 and CB-86 versions, but suggests that they are only appropriate for hard disk implementations because they consume 30% to 50% more disk space for the program code.

As a great lover of CB-80, I can hardly resist being offended by the

remark. To my observation, it is probably technically true that if you simply recompile source code written for the CBASIC compiler with the CB-80 compiler you will get a lot more code. This is only half the story, however. In my own experience, I have been able to reduce the total amount of code to about 10% LESS than the CBASIC compiler output by making some relatively minor struc-

I would suggest use of the CB-80 and CB-86 versions over the CBASIC ones.

tural changes. *Lifelines/The Software Magazine* has over the last year published several articles of mine that have told in detail how to do this. I won't repeat it here, but the strategy involves collecting the commonly used subroutines, converting them into external functions, and moving them to the root area.

Conclusions

This Univair Time Accounting system is well-organized, and, except for its lack of speed, should be regarded as a very good program which is likely to serve the purpose for which it is intended very well. I would suggest use of the CB-80 and CB-86 versions over the CBASIC ones. If an extra 50K of program code is going to sink the system, the computer is too small anyway.

CB-80 does many things (particularly sorting) much faster than CBASIC, and this would serve to mask the tedious file reorganizations. It is clear that on my 2.5 megabyte floppy system, the CBASIC version becomes hopelessly bogged down before it actually runs out of disk space.

Hopefully, Univair will act very soon to update this product to the state-of-the-art file management routines; but, in the meantime, the user can likely get satisfactory use out of this otherwise very credible product by avoiding the P-code versions, and by implementing the system on the highest performance disk system that he can afford.

SOFTWARE NOTES
WordStar 3.30

by Robert P. Van Natta

The latest from MicroPro International is **WordStar 3.30** for CP/M-80. Those of you who are worried about having to learn something new, however, may rest easy. The changes are few and far between.

The most visible change from Version 3.0 is that Version 3.3 displays a rather ugly MicroPro logo followed by a rather unfriendly message about how they are going to sue you if they get a chance.

Bugs fixed

Other changes are bug repairs. The new version does not include a repair list, so one can only guess at the exact number of bugs that have been fixed. Several that have been bothersome and seem to have faded are:

1) Version 3.0 had a habit of crashing if an attempt to mark a column wider than 80 characters was made. This appears to have been fixed.

2) The Block Delete command (↑KY) would delete material between the Block Markers (↑KB and ↑KK), but would leave the markers displayed. Now the markers are hidden by the delete. They may be turned back on by a ↑KH command.

3) Version 3.0 had problems with the spelling correction routine. This was described by various writers, including yours truly, as a bug in SpellStar, but the problem was actually within WordStar itself. Anyway, you can now use the spelling correction routine to search for Spellstar flags on large files without departing to never-never land.

Features added

There are a couple of new wrinkles worthy of note. The first thing that you will notice is that the installation program has been completely rewritten. The infamous WordStar Patcher no longer exists (or at least no obvious method of access to it is provided). Instead, many of the things that used to be adjusted with the patcher, such as the default help level and the default page formatting, are now menu driven. There is also a menu driven terminal installation routine for those terminals not on the list.

Documentation on adjusting the timing and on custom definitions for extra keys is not included. In fact, the documentation is completely rewritten in "IBM style" smallish slipcase binders, and all technical information has been eliminated.

Probably the most exciting thing added is a new feature in Mailmerge. Conditional printing is now supported in the 8-bit versions only. Conditional printing is a big phrase which means essentially that some important wisdom has been added to Mailmerge. Conditional printing is accomplished with a series of new "dot" commands which execute the logic of a BASIC IF. . .THEN statement. The familiar logical operators (=,<>,<,>,<=,=<,>=,=>) are all supported. Additionally, the IF condition may be compounded with AND or OR statements.

An example of a conditional statement might be as follows:

```
.av "Do you want to skip
   paragraph 1? (Y/N) ",flag,1
.if &flag& = "Y" .OR. &flag& = "y"
   GOTO nextsection
```

This text will be skipped if response to the .av question was either an upper or lower case "Y".

```
.ef nextsection
```

The logic of the NOT operator is also supported and, as a result, it is relatively easy to implement a routine which will sift a name and address file using any of the following criteria:

1) print those that are within range;
2) skip those are are within range;
3) print those that are out of range;
4) skip those that are out of range.

It is hard for me to really say whether these new commands are simple to use. The way I have it figured, however, if your level of computer literacy is such that you can write a ten line program in BASIC that will actually do something, you ought to be able to handle these new features. However, those who lack the basic literacy to program an IF. . .THEN statement or who can't figure out what GOTO means will never make it.

It is hard for me to really say whether these new commands are simple to use. The way I have it figured, however, if your level of computer literacy is such that you can write a ten-line program in BASIC that will actually do something, you ought to be able to handle these new features.

The manual presents some pretty good examples, and it is just possible that someone might get the conditional printing feature to work without ever understanding what an IF. . .THEN statement really is; but somehow I doubt it. In any event, even an experienced user is going to need to keep the manual close at hand because the syntax is exacting and unforgiving.

Bugs added

The new installation program for Version 3.3 is not without its problems. For example, there are menu Radio Shack installation choices for both Lifeboat and Pickles and Trout CP/M. In version 3.0 these choices implement the memory mapped display. Unfortunately, neither installation option actually works in Version 3.3. Rumor has it that MicroPro knows about this and will fix it some day; but since MicroPro has a policy of not telling anybody about their new versions, I wish everyone "good luck" in learning when the new version comes out.

You can get Version 3.3 to work on the Radio Shack equipment by selecting a terminal emulation routine. The SOROC 120 installation seems to work with LIFEBOAT CP/M, and it will work with ATON CP/M if you further modify the SOROC installation so as to NOT use the DELETE TO ENDLINE FUNCTION. You can get a working version under Pickles and Trout CP/M by putting the P&T terminal control codes in one at a time using the custom terminal option. However you measure it, though, it is a step backwards for the Shack equipment.

The special port driver routine for Radio Shack computers has been deleted from the installation program; however, the generic port driver routine is still there. It contains a bug, however. The manual and historic consistency both suggest that the port numbers be entered in 'hex'. It seems, however, that the input and output port numbers for Serial Port A on the Radio Shack computers are f5h and f7h respectively. The letter 'f' is rejected immediately as an illegal input character, making the entry of these port numbers impossible. You can, however, enter the decimal equivalents (245 and 247) and get the port driver to work. (You answer '4' to the other two questions about bits in the routine.)

It has always been my feeling that the port driver routine was a preferred method of driving a serial printer if you could get it to work for the reason that it bypasses the CP/M list device driver and I/O-byte toggle. If you don't have a lot of things plugged into the back of your computer, it probably doesn't make much difference; but if you are using a couple of different printers, you can drive one out through the LIST device and the other through the PORT DRIVER. You then brew up two versions of WordStar and put both on the same disk. You can then select which printer is going to get the output simply by choosing the appropriate version of WordStar. Similarly, if you have a serial printer that requires a software protocol, this is a good way to bypass any protocol that might be implemented in the operating system. It seems that it won't work with WordStar anyway; but if you disable it then nothing else will work with the printer. Implementation of the port driver will let you out of this dilemma.

Patching

For those interested in modifying the WordStar command structure or who desire to define special function keys in a way to do something useful, the method that I outlined for Version 3.0 in *Lifelines/The Software Magazine* (April, 1982) is still valid. Obviously, since the patcher is no longer available, you will have to use DDT for the patching. The addresses have changed ever so slightly, however. Accordingly it will take a little fiddling to figure out what is going on. The normal beginning of the keyboard patch area is now 0655h instead of 0649h. If you feel comfortable reading "hex dumps," identification of the patch areas of Version 3.3 won't stop you for long, as they are very similar to the earlier versions.

If you want to spruce up WordStar 3.3 a little bit, I would suggest patching the following addresses to 00h: 02b2h, 3f1dh, and 411fh. Location 02b2h is what used to be known as label DEL4. It is factory issued with a 40h at that address. This regulates the amount of time that you must stare at the MicroPro Logo and trade secret notice each time you load WordStar. Patching the other two addresses will suppress the display of the trade secret notice and logo respectively.

I suppose that MicroPro could be offended by my telling the world how to suppress the display of their perfectly ugly logo and their very unfriendly threat to sue their enemies. I informally understand that the stuff was hung on the front of WordStar in a transparent effort to prop up their legal efforts against the software rental companies. Before you suppress the display of these items, you surely should memorize the contents of MicroPro's page-long threat to sue you. Likewise, if you think that you might forget its contents, copy it down on a sheet of paper and tape it to the front of your computer for future reference.

SpellStar

Those of you who are regular readers already know that I have reviewed several versions of SpellStar over the last couple of years. Suffice it to say that SpellStar has a reputation for being slow and buggy. The speed has been improved some from the early versions and, at last, the obvious bugs have disappeared.

I suppose it would be faint praise to say that the main difference between SpellStar Version 3.3 and the earlier versions is that this version works. So be it!

I suppose it would be faint praise to say that the main difference between SpellStar Version 3.3 and the earlier versions is that this version works. So be it!

Conclusions

The conditional printing routine in Mailmerge is the only thing that is really new in this whole system. For IBM-PC users there is memory mapping, but no conditional printing. There is new documentation for everyone. On the balance, however, for current users, the new versions is mostly a yawn. I paid the $85 for the WordStar update and $25 a pop for SpellStar and Mailmerge. Frankly, mine are going back on the shelf, at least until I can find a patch that will deal with the failure of Version 3.3 to support the memory mapping facilities of my Radio Shack computer. It is perhaps a sign of the times that one of the strong points of the 16-bit version is the addition of memory-mapping for the IBM-PC.

Users' Group Corner

Editors Note: We hope you will write in and give us information about your users group or computer club. Our Users' Group Corner is designed to help you find computer clubs in your area or new clubs that your existing club can exchange information with.

CPMUG
1651 Third Avenue
New York, NY 10028

The complete CPMUG™ catalog, which consists of nearly 100 volumes, is available for $10, prepaid to the United States, Canada, and Mexico; $15, prepaid to all other countries. (All checks must be in U.S. currency drawn on a U.S. bank.)

Software in the library, obtainable exclusively on diskettes, is available for a prepaid media and handling charge, as follows:

FORMAT	DESTINATION
8" IBM	U.S., Canada, Mexico—$13
8" IBM	All other destinations—$17
North Star/Apple	U.S., Canada, Mexico—$18
North Star/Apple	All other destinations—$21
KAYPRO II	Same price as Apple II.

PLEASE CLEARLY SPECIFY THE FORMAT YOU WANT WITH YOUR ORDER

This payment covers the cost of the diskette(s), packaging, and postage. Domestic shipping is via UPS where a full street address is given; all other orders are via U.S. Postal Service. For more information, call (212) 860-0300, ext. 343.

TRACE
PO Box 6922
Station "A'
Toronto, Ontario M5W 1X6
CANADA

TRACE is the Toronto Region Association of Computer Enthusiasts, which is a non-profit organization of people who are interested in personal computing. TRACE general meetings are held on the fourth Friday of the month in Room J206, at Humber College in Rexdale, at 8 p.m. Membership is $20 a year, which includes a subscription to the TRACE newsletter.

The Boston Computer Society
Three Center Plaza
Boston, MA 02108

The Boston Computer Society is the largest nonprofit personal computer association in the US. Their goal is to help computer users and people who want to know what a computer can do for them. The BCS has general meetings, and they have many users' groups and also special interest groups. Membership is $20 a year which includes a subscription to *Computer Update* and a subscription to the Society's *CALENDAR* newsletter.

(continued from page 2)

"integrated."

It's not unlike going to Sears and buying a combination chainsaw, screwdriver and hammer. The more combinations, the higher the probability that the greatest benefit in the case of software may be to take the diskettes, reformat them and use them for something useful. In the case of hardware . . . perhaps boat anchors!

Recognize the fact that the basic premise for many of the new offerings is that people of corporate America are desperate to spend their whole day locked in their offices reading electronic mail, creating spread sheets, playing with mice, generating pie charts, and engaging in data base management. Not to mention viewing fifteen documents simultaneously. And then, of course, there is networking. When was the last time you called someone in corporate America and found them in their office for any reason? Thus the basis for suggesting that every device should be connected to every other device!

An interesting concept — networking — people love to talk about it incessantly but who do you know that uses their micro in a network in any significant way? You got it . . . another important contribution to the field of ether-ware!

Flash! TI has now formally challenged Timex for the title of who has more closet shelf space in American homes. The demise of the TI 99/4 is indeed sad in view of the fact that the machine has excellent color graphics and speech capability. By the way those of you wondering what to do with the 99/4s might consider adding the speech module and using them for voice synthesis on your home systems.

The industry is increasing its pace as more and more emphasis is put on software. Doomsday sayers will undoubtedly continue with their dire and dark predictions.

"But fear not, the game's not lost; there's much afoot . . . and the best is still to be."

PRODUCT STATUS REPORTS

New Products

UTL

EWDP Software, Inc.
PO Box 40283
Indianapolis, IN 46240
(317) 872-8799

UTL is a universal utility for use on any CP/M system. It consolidates six commonly used file functions into a single program with a concise menu. All commands are accessed from the menu with a single key stroke, are self-typing, and provide extensive operator prompting. There is absolutely no command syntax to remember. Each function has extensions and features not generally available with other utilities. The directory command has full selectivity, an extremely fast recursive sorting algorithm and an optimized columnar display with file sizes and attributes. It provides remaining disk space available even on selective directories while automatically accounting for the different techniques required for implementation under CP/M 2.2 and bank-switched CP/M-Plus systems. The TYPE command is fully buffered, has screen pauses and features and exclusive bidirectional capability to display the next screen or previous screens from a 10,000 character buffer. UTL is written in optimized assembly language for extremely fast and efficient operation and will run on any CP/M microcomputer. No installation procedure is required.

Requirements: CP/M, Z/80

Price: $29.95

1040 PLUS

1040 PLUS
1953 East Apache Boulevard
Tempe, AZ 85281
(602) 968-3350

This tax package calculates and prints income tax returns. The formatting and instructional procedures in this program were created with the novice user in mind. It is menu-driven. It contains all the schedules and 12 of the most frequently used forms including: 1040, 1040Am 1040X, 1040-ES, A, B, C, D, E, F, G, R/RP, SE, W, 2106, 2119, 2110, 2441, 3468, 3903, 4562, 4625, 4684, 4797, 5695, and 6252.

Requirements: CP/M 2.2, MS-DOS or PC-DOS
Price: N/A

DOCUPOWER!

Computing!
2519 Greenwich
San Francisco, CA 94123
(415) 567-1634

This program is an organizer that works with any word processor. It assembles random paragraphs, sections, pages, or any other word processor texts into a master INDEXED resource file of reusable ideas. By marking pages or paragraphs with Docupower! marks, Docupower! will automatically add these pages and paragraphs to a master resource file and make an index sorted by category for you. You make up the new text by picking numbers from this indexed catagory file. It remembers the series of numbers and creates the new text file for you anytime you want to print or edit.

Requirements: CP/M-80, CP/M-86, PC-DOS
Price: $149

New Books

BASIC Programming Primer, Second Edition by Mitchell Waite and Michael Pardee

Howard W. Sams & Co., Inc.
4300 West 62nd Street
Indianapolis, IN 46268

Directed at beginners, the text has been expanded to include Microsoft BASIC for the IBM-PC, 16-bit BASIC statements and more. The self-instructional format includes self-tests and answers. There are working examples and game-style program examples. A BASIC language reference card has been bound into the book, but may be easily removed for handy reference. Tips and techniques for professional-style program planning and coding are offered throughout the text. Price: $17.95

PROGRAMMING A PERSONAL COMPUTER by Per Brinch Hansen

Prentice-Hall, Inc.
Englewood Cliffs, NJ 07632
(201) 592-2158

Hansen offers a software system powerful enough to support development of nontrivial programs on a personal computer, yet simple enough to be studied in detail at all levels of programming. This system is his Edison system, supporting development of programs written in the Edison language, a Pascal-like language developed for microprocessors. Hansen explains how the Edison programming language was developed and implemented, including both the operating system and compiler program text.

Price: $18.95

New Versions

COBOL-80	v.4.66
CP/M WorkShop	v.2.0
FORMULA-II for CP/M-80,86; PC/MS-DOS	v.2.2
GrafTalk for CP/M-80 and PC-DOS	v.2.15
HALO w/HERCULES board for PC-DOS only	v.1.85
LATTICE C COMPILER available for either MS-DOS 1.x or MS-DOS 2.x	v.2.0
MAG/base-1,2,3 compiled under CB80	v.3.2
MAGICBIND	v.1.11
MAGICPRINT	v.2.0
MAIL-COM PC/XT	v.1.3
MATH★	v.3.043c
MICROSPELL-PC	v.5.0
PANEL-86	v.5.01
PL/I-80	v.1.4
Precision BASIC for MS-DOS	v.1.7
Precision BASIC for CP/M-80 (Z80)	v.1.2
ResQ for PC-DOS only	v.1.1
SMARTKEY-II for CP/M-80 and MS-DOS	v.1.0
SMARTPRINT	v.1.0

LIFEBOAT™ Associates:
The full support software source.

Reach for the programming horizon of the 80's with Lattice™ C, the fastest C compiler.

Set the course of your next software project towards greater power and portability by selecting Lattice C, recognized as the finest and fastest 16-bit C compiler. Lattice C, the full implementation of Kernighan and Ritchie, continues to lead the way by announcing full memory management up to 1 Megabyte. Major software houses including Microsoft, MicroPro and Sorcim are using Lattice C to develop their programs.

Lattice C is available for a wide variety of 16-bit personal computers including IBM®, NCR®, Texas Instruments, Victor, Wang and other microcomputers running PC™-DOS, MS™-DOS and CP/M86™.

Call LIFEBOAT at 212-860-0300 for free information on the Lattice C family of software development tools.

Lifeboat offers Lattice C with a tested set of software tools to provide a complete development system including:

HALO™
PANEL™
PMATE™
PLINK™-86
C-FOOD SMORGASBORD™
LATTICE WINDOW™
FLOAT87™

Color graphic primatives
Screen design aid
Customizable program editor
Overlay linkage editor
Screen and I/O utilities
Multi-window functions
8087 floating point math

LIFEBOAT
Associates

1651 Third Avenue
New York, NY 10028
212-860-0300

Please send me free information on:

- ☐ Lattice and development tools
- ☐ How to get your software published
- ☐ Corporate purchase program
- ☐ Dealer program
- ☐ OEM agreements

☐ Send me the complete LIFEBOAT software catalog. $3.00 enclosed for postage and handling.

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

Telephone _____

LATTICE, C-FOOD SMORGASBORD and LATTICE WINDOW, TM Lattice, Inc.

LIFEBOAT, TM Lifeboat Associates
HALO, TM Media Cybernetics
PANEL, TM Roundhill Computer, Ltd
PMATE and PLINK, TM Phoenix Software

FLOAT87, TM Microfloat
IBM and PC, * TM International Business Machines
MS, TM Microsoft
CP/M86, TM Digital Research

Lifelines™/ The Software Magazine™
1651 Third Ave., New York, New York 10028

Second Clas
At New